제2권 통권5호 1989년4월1일 발행(매월 1일 발행) 1988년10월10일 등록(등록번호 라-3794) ㉾140-150 서울특별시 용산구 갈월동 17

퍼스널컴퓨터
하이테크

89/04

특집/PC통신이란 무엇인가
/RS-232C란/접속상태/케이블/통신속도
/통신방식/송신부호/동기방식/

화일의 결합과 분리/ 간단한 방법으로 키정의를 바꾼다/메모리 할당과 할당 해제
360KB 드라이브로 360KB이상의 데이터를 복사하려면/QUICKEY.COM의 분석/
미분이 가능한 계산기 프로그램



새로 선보인 한글 워드프로세서 'ᄒᆞᆫ글'
완벽하게 회로를 디자인해주는 만능 PCB용 캐드
DASOFT-16

□ 편집자 노트 □

생동의 계절 4월을 맞아 새롭게 단장하고자 합니다.

최근 들어 거리의 풍경들이 하루가 다르게 변해 갑니다. 나뭇가지마다 파릇파릇 새싹이 움트고 겨울동안의 긴 잠에서 깨어나 묵은 먼지를 털고 새 단장을 하는 등 지난 겨울의 우중충한 회색의 거리와는 달리 새로운 모습입니다.
보는 이로 하여금 항상 싱그러움과 싱싱한 젊음을 느끼게 하여 사뭇 가슴이 설레기까지 합니다.
싱싱한 젊음이란 늘 생동감이 넘치며, 때가 묻지 않은 순수를 간직할 수 있다는 의미에서 언제 보아도 새롭습니다.
4월은 그만큼 모든 사물을 바뀌게끔 할 수 있는 활기찬 계절이라 말할 수 있읍니다.
만물에서 항상 새롭게 소생하는 싱그러움을 느낄 수 있고, 이러한 싱싱함은 누가 보아도 아름답게 느껴집니다.
생동의 계절 4월을 맞아 저희 잡지도 새롭게 단장을 하고자 합니다.
우선 컴퓨터를 접하는 대부분의 사람들이 한글 워드프로세서를 사용하고 있으면서도 매뉴얼 등의 부실로 인하여 제대로 활용하지 못하는 점을 감안하여 국내에서 사용되고 있는 각종 워드프로세서 (예를 들어 보석글, 팔란티어 워드프로세서, 한글 2000 등)를 하나씩 분석하여 실무자들이 이를 토대로 응용할 수 있도록 하고자 합니다. 이러한 작업은 유저들로 하여금 워드프로세서를 올바로 사용할 수 있게 해주고 각자 업무에 맞게 응용할 수 있어 많은 도움이 되리라 생각합니다.
그리고 최근 들어 컴퓨터가 대중화되면서 업무용으로 많이 사용되는 추세입니다. 저희 잡지에서는 이러한 점을 감안하여 업무용 간이언어인 디베이스와 로터스 등 실제 업무에 필요한 패키지 강좌를 확충하여 컴퓨터를 전공하지 않고 사회에서 처음 대하는 사람들도 쉽게 이해할 수 있으며, 실무에 도움이 되게끔 하겠읍니다.
이밖에도 컴퓨터를 빨리 익히고 활용하기 위해서 반드시 필요한 것은 다양한 유틸리티의 사용입니다. 따라서 이러한 각종 유틸리티들을 분야별로 모아 분석하고 해설하여 독자로 하여금 폭넓게 이해할 수 있도록 하여 컴퓨터를 제대로 활용할 수 있게끔 하겠읍니다.
실제로 우리 주위를 보면 컴퓨터를 제대로 다루지 못한다는 말이나 활용을 하려고 하는데 어떻게 사용해야 되는지 모르겠다 등의 생각을 가지고 있는 사람들이 너무나도 많습니다.
저희 잡지의 역할은 하루빨리 이 땅에서 이러한 컴퓨터 문맹자를 깨우쳐 누구나가 생활속에 컴퓨터를 다룰 수 있는 정보화사회를 만드는 일이라고 생각합니다.
앞으로 많은 부분을 새롭게 꾸며 항상 생동하는 싱싱한 4월의 잡지를 만들고자 노력하겠읍니다.

MS-DOS입문 [3]
COMMAND.COM과 외부명령

박승제/여주컴퓨터학원장

CONFIG.SYS의 작성

DOS 디스크에는 확장자가 .SYS인 화일이 있다. 이 .SYS 화일은 컴퓨터가 처음 켜질 때 도스가 읽어들이는 화일이라고 말한 바 있다. 이 화일은 사용자가 관여할 수 없도록 되어있지만 예외의 화일이 하나 있는데 그것이 바로 CONFIG.SYS이다.

CONFIG.SYS는 컴퓨터가 켜질 때 읽어들이는 .SYS화일을 지정하는 기능과 버퍼의 수, 한번에 다룰 수 있는 화일의 수 등을 지정하는 역할을 한다. 이중에서도 .SYS화일을 지정하는 역할은 매우 중요하다. 만약 CONFIG.SYS에서 .SYS화일을 지정하지 않는다면 .SYS 화일은 실행되지 않을 것이며, 그 컴퓨터의 한글체계가 'HCON.SYS' 등과 같이 .SYS화일로 지정되어 있는 기종(행정 전산망용 2바이트 완성형 한글, 이른바 KS 한글)에서는 한글을 사용할 수 없게 된다.

CONFIG.SYS는 에디터에서도 작성이 가능하지만 보통은 배치화일을 작성하는 것처럼 'CONsole'에서 하게 된다. CONFIG.SYS의 작성 예는 다음과 같다.

```
COPY CON CONFIG.SYS
DEVICE=ANSI.SYS
FILES=15
BUFFERS=10
[Ctrl]+[Z]
```

DEVICE는 원래 장치명을 뜻하지만 여기서는 읽어들일 .SYS 화일명을 기술하는 명령이다. 이 예에서는 DEVICE가 읽어들일 .SYS 화일명으로 ANSI.SYS를 지정하였다. 그러나 KS한글을 사용하는 기종인 경우에는 ANSI.SYS 대신 'HCON.SYS'를 지정하면 될 것이다.

DEVICE를 여러개 지정할 경우는 여러번 기술하면 된다. 즉 ANSI.SYS와 HCON.SYS를 모두 읽어들일 경우에는 다음과 같이 입력한다.

```
DEVICE=ANSI.SYS
DEVICE=HCON.SYS
```

FILES는 프로그램상에서 오픈할 수 있는 화일의 최대 수를 설정하는 명령이다. 초보자의 경우에는 따로 설정할 필요가 없지만 응용 프로그램에 따라서는 FILES를 설정해야 하는 경우도 있다. 예를 들어 DBASE Ⅲ 등의 데이터베이스 프로그램에서는 15개로 설정하도록 되어 있다. FILES를 설정하지 않으면 자동적으로 5개로 설정된다.

BUFFERS는 입출력용 버퍼를 할당하는 명령이다. 버퍼는 디스크나 프린터로 데이터를 입출력할 때 임시 기억장소로 이용된다. 예를 들어 프린터로 출력하는 경우를 생각해 보자. 컴퓨터는 프린터로 데이터를 보낼 때 직접 글자를 한자씩 보내는 것이 아니라 프린터에 보낼 문자들을 일단 버퍼에 한자씩 보내 저장하게 하고 버퍼가 가득차면 버퍼의 내용을 프린터로 한꺼번에 전송하게 되는 것이다.

디스크의 경우도 마찬가지이다. 즉 디스크에서 정보를 읽어들일 경우에도 1 버퍼분을 한번에 가져온다. 그 다음엔 버퍼의 내용을 하나씩 꺼내서 처리하게 된다. 이렇게하면 매번 디스크를 액세스하지 않아도 되므로 입출력 시간이 빨라지게 된다.

여러개의 화일을 자주 액세스하는 프로그램의 경우에는 버퍼의 수를 많이 잡아두면 입출력 속도가 빨라진다. 그러나 자주 액세스하지 않는 프로그램의 경우에는 사용자의 메모리만 줄어들 뿐 별다른 이점은 없다. BUFFER의 수는 별도로 지정하지 않는한 자동적으로 2개가 된다.

컴퓨터의 부팅과정

DOS디스켓은 대략 30~50개의 화일로 구성되어 있다. 이 중에서 가장 중요한 것이 COMMAND.COM이라는 화일인데 이것은 우리가 내린 명령을 해석하여 실행하는 역할을 하는 것으로 흔히 DOS라고 하면 COMMAND.COM을 일컫는 경우가 많다.

DOS디스크에는 COMMAND.COM 말고도 중요한 화일이 2개 더 있는데 이것은 'IBMBIO.COM'과 'IBMDOS.COM'이라는 것이다. (이들 화일을 IO.SYS, DOS.SYS로 표기하는 DOS도 있다.)

이 화일들은 'DIR'명령으로는 화면상에 나타나지 않으며, 'DEL' 명령으로도 지워지지 않는다. 또한 COPY 명령으로 복사할 수도 없다. DOS에서는 이들 두 화일을 '숨겨진 화일(Hidden

File)', 혹은 '시스템 화일(System File)'이라고 부른다.

컴퓨터가 처음 켜지면 ROM에 있는 프로그램에 의해 컴퓨터는 하드웨어를 체크한다. 즉 사용 가능한 메모리는 얼마나 되며, RAM이 제대로 작동하고 있는지 등을 체크한 다음 디스크의 '부트 스트랩 로더'를 읽어들인다.

부트 스트랩 로더(Bootstrap Loader)는 흔히 '부트 레코드'라고도 부르는데 이는 디스켓의 맨 첫머리 부분이다. 이 부트 스트랩 로더는 디스켓을 포매팅할 때 작성되는데 주로 그 디스켓이 어떤 형식으로 포맷되었으며 그 디스크에 시스템용 화일이 있는지의 여부가 기록되어 있다. 이를테면 그 디스켓이 단면 단밀도인지, 양면 배밀도인지, 또 트랙당 몇개의 섹터를 가지고 있는지를 표시한다.

그 디스크에 시스템 화일이 기록되어 있는 경우에는 그들의 화일명과 DOS의 버전이 기록되게 된다. 디스크를 포맷하지 않으면 읽어들이지 못하는 것은 그 디스크에는 아직 부트 스트랩 로더가 작성되지 않았기 때문이다.

컴퓨터는 부트 스트랩 로더를 읽어서 시스템 화일이 있는지를 확인한다. 만약 시스템용 화일이 없다면 다음과 같은 메시지를 출력한다.

```
Non system disk or disk error
Replace and strike any key when ready
```

시스템 화일이 존재하면 메모리의 맨 마지막부분에서 IBMBIO.COM을 읽어들인다. IBMBIO.COM은 드라이브나 프린터 등의 주변장치에 대한 초기화를 실행한다. 또한 드라이브는 몇대인지, 종류는 어떤 것인지, 통신용 포트 등의 주변장치 등은 장착되어 있는지의 여부를 조사한다.

조사가 끝나면 다음에는 MSDOS.COM을 읽어들이게 된다. MSDOS.COM은 주변 장치에 대한 작업영역을 확보한다. 여기까지 수행되었으면 디스크에 CONFIG.SYS가 있는지를 조사해서 존재하면 읽어들인다.

그 다음에는 COMMAND.COM을 읽어들인다. 그 다음은 프롬프트 'A:\>'를 출력하고 사용자의 입력을 기다리게 된다.

여기까지의 과정에서 컴퓨터가 장착한 메모리중의 일부분이 시스템에 할당된다. 또한 COMMAND.COM도 일종의 프로그램이므로 당연히 메모리상에 위치한다. 이들의 메모리는 사용자의 영역에서 제외되므로 사용자의 메모리는 그만큼 줄어들게 된다.

내부 · 외부 코맨드

앞서 설명했듯이 COMMAND.COM은 사용자가 내린 명령을 해석하여 실행하거나 지정한 화일을 로드하여 실행하는 역할을 한다. COMMAND.COM의 기능이 다양할수록 COMMAND. COM의 크기는 커지게 된다. 즉 자세히 설명하면 할수록 말이 많아지게 되는 이치와 같다. 따라서 이런 저런 잡다한 기능까지 다 고려하여 COMMAND. COM을 만들게 되면 COMMAND.COM의 사이즈는 커지고 상대적으로 사용자가 사용할 영역은 줄어들기 마련이다. 또한 COMMAND.COM이 커지면 처음 부팅할 때 더 많은 시간이 걸리게 된다.

이와 같은 이유 때문에 우리가 사용하고 있는 DOS를 설계할 때 COMMAND. COM에는 가장 빈번하게 이용되는 기능만을 처리하도록 하여 그 사이즈를 최소한으로 줄여두었다. 그리고 COMMAND.COM으로 처리하지 못하는 명령들은 각각 별개의 프로그램으로 작성하여 필요시 그 프로그램을 실행함으로써 일련의 처리를 하도록 되어 있다.

이를테면 디스켓을 포매팅하는 기능은 COMMAND.COM이 하는 것이 아니고 FORMAT.COM이라는 프로그램이 하게 된다. 그러나 화일의 카피, 삭제와 같은 기능은 COMMAND.COM이 직접 처리한다. DOS상에는 COMMAND. COM이 처리하는 명령을 '내부 명령어', FORMAT.COM처럼 별개의 화일로 작성되어 있는 것을 '외부 명령어'라고 구분하여 부른다.

DOS디스켓에 수록되어 있는 대부분의 .COM화일이나 .EXE 화일들이 바로 이 외부 명령에 속한다. 외부 명령어들은 모두 별개의 화일로 수록되어 있으므로 그 기능을 실행하려면 반드시 디스크 상에 그 화일이 존재하여야 한다. 이번 호에는 외부 명령어의 기능에 대하여 알아보기로 한다.

ASSIGN.COM

[형식] ASSIGN D1=D2

이 ASSIGN.COM은 드라이브를 강제로 지정하는 역할을 하는 것으로 [형식]에서 D는 A, B, C 등과 같은 드라이브 지정문자이고, 내용은 D1 드라이브를 D2로 바꾸라는 것이다.

이를테면 하드디스크의 사용자라면 B:드라이브를 사용하는 경우가 별로 없을 것이다.(왜냐하면 B:는 플로피디스크 드라이브를 지정하는 문자이기 때문이다.) 그러나 응용 프로그램에서 빈번하게 B:드라이브를 액세스하려고 한다면 곤란하게 된다. 이때는 'ASSIGN B=C'로 설정해두면 응용 프로그램에서 B:를 액세스할 경우에는 B:대신 자동적으로 C:를 액세스하게 된다.

실제로 'ASSIGN B=A'라고 입력해보라. DIR B:의 명령에 A:드라이브의 내용이 출력될 것이다. 그렇게 되면 B:드라이브는 사실상 사용 불가능한 상태가 된다.

플로피디스크 드라이브를 사용하는 프로그램에서는 대부분 프로그램 자체는 A:드라이브에, 그리고 데이터는 B:에 쓰도록 설정된다. 이런 프로그램을 하드디스크에서 사용할 경우에는 프로그램이든 데이터이든 모든 C:를 이용해야 할 것이다. 이때는 프로그램 자체를 수정하지 않고도 'ASSIGN B=C'

라고 입력함으로써 프로그램이 수정된 효과를 거둘 수 있다.

ASSIGN 명령에서는 드라이브의 지정에 ':'을 붙이지 않는다는 점에 유의하기 바란다. 그리고 ASSIGN만 다시 입력하면 드라이브의 지정이 원래대로 복원된다.

ATTRIB.COM

ATTRIB.COM을 사용하기 위해서 우리는 먼저 화일의 속성(Attribute)에 대해서 이해해야 한다. 속성은 그 화일의 성격을 나타낸다. DOS에서는 화일의 속성을 다음과 같이 구분한다.

① 보통 상태(normal)
② 읽기전용(Read Only)
③ 숨겨진 화일(Hidden File)
④ 시스템 화일(System File)
⑤ 백업된 화일(Archived File)

우리가 DIR명령으로 보는 대부분의 화일은 ①의 속성을 갖는다. 이 보통상태의 화일은 사용자의 명령에 의해 지워지기도 하고, 화일명을 변경할 수도 있다.

②의 속성은 화일을 읽거나 실행할 수는 있지만 삭제하거나 변경하는 것이 금지된 화일임을 나타낸다.

③의 속성을 갖는 화일들은 디렉토리상에는 나타나지도 않으며 삭제, 복사, 이름 변경 등도 할 수 없다. 그러나 DOS상에서 실행시키는 것은 가능하다.

④는 앞에서 설명한 IBMBIO.COM과 IBMDOS.COM에 부여된 속성이다.

⑤는 뒤에서 설명할 BACKUP.COM에 의해 복사된 적이 있는 화일임을 나타낸다.

ATTRIB.COM은 이 5개의 속성중 ②를 설정하거나 해제할 수 있다. 그 외의 속성을 변경하는 기능은 DOS에는 없으므로 'PCTOOLS'이나 '노턴 유틸리티' 등의 프로그램을 이용하면 가능하다.

이 ATTRIB.COM의 형식은 다음과 같다.

[형식] ATTRIB [+R/-R] [드라이브:] [경로] 〈화일명〉

여기서 [ ]로 나타낸 부분은 생략해도 되고, / 는 선택을 의미한다. 그리고 +R은 화일의 읽기전용을 지시하고 -R은 읽기전용을 해제한다는 의미이다. 또한 [ ]부분을 생략하고 ATTRIB만을 사용하는 경우에는 화일의 속성을 나타낸다.

(예) ATTRIB +R A:COMMAND.COM

이 예는 COMMAND.COM을 읽기전용으로 속성을 변경하는 보기이다. 화일명에 와일드카드 문자('*', '?')를 사용할 수 있으므로 여러개의 화일의 속성을 한꺼번에 변경할 수도 있다.

속성이 읽기전용으로 설정된 화일은 DEL 명령에 의해서 삭제되지 않으므로 귀중한 자료의 경우에는 화일의 속성을 읽기전용으로 해두면 불의의 사고에도 화일을 보호할 수가 있다.

ATTRIB A:*.*에 의해 모든 화일의 속성을 화면에 표시한다. 화일의 속성은 R(읽기 전용), A(백업된적 있음) 등으로 표시한다. 읽기전용의 화일을 삭제하려는 경우에는 'Access Denied' 이라는 메시지를 출력한다.

BACKUP.COM과 RESTORE.COM

디스크(하드나 플로피)에 저장된 자료는 전기적인 충격(작은 정전)이나 자기, 혹은 물리적인 자극에 의해 데이터가 손상당할 가능성을 항상 가지고 있다.

손상당한 화일이 프로그램인 경우는 그 프로그램을 구입한 곳이나 다른 경로를 통해 다시 입수하면 되지만 손상당한 화일이 데이터인 경우는 원상태로 복구하지 못하면 매우 심각한 상황에 부딪치게 된다. 잘못하면 수년간 축적해둔 데이터를 일시에 날려버릴 수도 있다.

이 경우에는 수년전의 시점으로 돌아가 데이터를 처음부터 다시 입력하지 않는다면 그 회사의 업무는 스톱될 것이 분명하다. 그렇기 때문에 컴퓨터의 사용자는 프로그램보다는 데이터 관리에 더욱 신경을 써야 한다.

백업(Backup)은 화일이나 데이터를 불의의 사고로부터 보호하기 위해 다른 기억 매체(디스크나 테이프)에 복사(Copy)해두는 것을 의미한다.

불의의 사고로부터 화일이 깨지는 경우를 대비하는 방법은 화일을 주기적으로 다른 디스크에 복사(Backup)하여 두는 길밖에 없다. 이런 복사본을 가지고 있는 경우에는 화일이 깨지는 경우라도 복사본으로부터 다시 카피를 해오면 복사본을 만들어둔 시점까지는 복구가 되기 때문이다.

이런 것은 컴퓨터를 다루는 사람에게는 상식적인 일이지만 수많은 화일을 일일이 백업한다는 것은 매우 번거로운 일이며, 또한 백업을 하다보면 예전에 백업한 후로 단 한번도 수정한 적이 없는 화일들이 다시 백업되기도 한다.

이러한 수고를 덜어주는 화일이 있는데 바로 BACKUP.COM이 그것이다. 이 화일은 백업을 하되 몇가지의 옵션을 주어 그 옵션에 해당하는 화일만 골라서 백업하도록 하는 아주 유용한 프로그램이다.

BACKUP.COM의 형식과 옵션들을 열거하면 다음과 같다.

[형식] BACKUP [D1:]화일명 [D2:] 화일명 /옵션/옵션....

여기서 D1은 자료가 있는 드라이브이고, D2는 백업받을 드라이브이다.

[옵션]

/S : 서브 디렉토리의 화일들도 모두 백업한다.

/M : 마지막 백업한 이후의 화일들만 백업한다.

/D:mm-dd-yy : 지정한 날짜(mm-dd-yy)이후 작성된 화일만 백업한다.

/T:xx-yy-zz : 지정한 시간(xx : yy : zz) 이후에 작성된 화일만 백업한다.

/A: 백업받을 디스크에 이미 화일이 존재할 때 그 화일에 더해져서 백업된다. 이 옵션을 생략하면 백업받을 디스크에 있는 화일들은 모두 지워진다.

/P: 백업할 디스크의 화일이 많을 경우 서브 디렉토리를 만들어 백업한다.(플로피디스크에 수록할 수 있는 화일의 최대 갯수는 112개 임을 상기하라)

생략시: 커런트 디렉토리의 모든 화일이 백업된다.

이 옵션을 적절하게 이용하면 백업하는 시간을 최소한으로 줄일 수 있다. 예를 들어 1989년 2월 20일에 백업을 해 둔 적이 있었다면 현 시점에서의 백업은 그 날자 이후에 작성된 것만 지정하여 백업하면 될 것이다. 왜냐하면 화일을 수정하면 그 화일이 만들어진 날자 역시 수정된 날자로 바뀌기 때문이다.

예를 들면 다음과 같다.

```
BACKUP A:*.DAT B:/D:2-29-89 /A
```

이 보기는 A:디스켓의 확장자가 .DAT인 화일 중에서 89년 2월 29일 이후에 작성된 것만 골라 B:로 백업하되 B:디스켓에 다른 이름의 화일이 존재하면 그 화일들을 지우지 말고 그 다음 영역에 기록하라는 의미이다.

이때 백업받을 디스크는 포맷이 되어 있어야 한다. 그리고 '/A' 옵션을 생략하면 B:디스켓의 모든 화일이 지워지게 되므로 '/A' 옵션을 생략할 경우에는 주의해야 한다.

화일이 손상을 입어 사용불가능할 경우 위의 BACKUP 코맨드로 백업해 둔 경우에는 RESTORE.COM으로 복구할 수 있다. RESTORE.COM의 사용형식과 옵션은 다음과 같다.

[형식] RESTORE [D1:] [D2:] /옵션/옵션.....

D1은 백업된 화일을 가지고 있는 드라이브이고, D2는 복구할 드라이브이다.

[옵션]

/S: 지정한 서브 디렉토리를 모두 저장시킨다.

/P: 숨겨진 화일이나 읽기전용 화일은 복구 전에 프롬프트를 출력하여 복구할지의 여부를 결정한다.

/B:mm-dd-yy: 지정한 날짜 이전에 마지막으로 변경된 화일만 복구한다.

/A:mm-dd-yy: 지정한 날짜 이후에 마지막으로 변경된 화일만 복구한다.

/E:xx:yy:zz: 지정한 시간 이전에 마지막으로 변경된 화일만 복구한다.

/L:xx:yy:zz: 지정한 시간 이후에 마지막으로 변경된 화일만 복구한다.

/M: 마지막 백업 이후 변경된 화일만 복구한다.

/N: 복구할 드라이브에 없는 화일만을 복사해온다.

다음은 RESTORE.COM으로 복구한 예이다.

(예) RESTORE B: A: /A:2-1-89

JOIN.EXE

[형식] JOIN [D1: D2\서브 디렉토리명] [/D]

이것은 D1으로 지정한 드라이브를 D2 드라이브의 서브 디렉토리로 할당하라는 내용이다.

JOIN.EXE는 드라이브를 서브디렉토리로 취급하도록 하는 유틸리티이다. 이를테면 B:드라이브를 A:드라이브의 어느 서브디렉토리로 취급하도록 한다는 뜻이다.

하드디스크를 사용하는 사람들은 여러개의 서브디렉토리를 만들어 사용한다. 이때 프로그램에서 생성된 데이터는 그 프로그램이 있는 서브디렉토리와는 다른 디렉토리에 저장될 것이다. 이런 프로그램을 2대의 플로피드라이브로 수행하려는 경우에는 그 프로그램의 데이터를 저장하는 서브디렉토리를 지정한 부분을 수정해야 할 것이다. JOIN.EXE는 이 경우에 드라이브를 서브디렉토리와 같이 취급하므로 프로그램을 수정하지 않아도 하드 드라이브에서 동작하는 프로그램을 플로피 드라이브에서 수행할 수 있도록 해준다. 또한 JOIN.EXE는 하드 드라이브를 사용하는 경우 플로피 드라이브를 JOIN함으로써 그 드라이브를 하드 드라이브의 서브디렉토리중의 하나로 취급하는 것이 가능하다. 이때 JOIN으로 지정된 드라이브를 A:, B: 등과 같이 지정하면 에러가 발생한다.

(예) JOIN B: A:\TEST

이 예는 B:드라이브를 A:드라이브의 TEST 디렉토리로 지정한 것이다. 이렇게 지정해두면 DIR A:\TEST\의 명령에 B:드라이브의 내용이 출력된다. 이때 'TEST' 디렉토리는 비어있어야 하며, JOIN으로 지정할 당시 A:에 'TEST'디렉토리가 없으면 저절로 만들어진다.

이 경우 지정한 JOIN을 해제하려면 JOIN A:/D ↵ 를 입력해야 한다. JOIN만 입력하면 현재의 JOIN 상태를 표시해 주기 때문이다.

SUBST.EXE

[형식] SUBST D1: [D2:] [서브디렉토리명] [/D]

SUBST.EXE는 특정한 서브디렉토리를 가상의 드라이브로 취급해준다. 하드 드라이브에서 서브디렉토리를 깊게 만들었을 경우 일일이 그 경로를 지정하는 것은 상당히 번거로운 일이다. 이때 SUBST.EXE로 그 디렉토리를 하나의 가상드라이브, 이를테면 D:와 같이 지정해두면 그 다음엔 D:만 지정하면 자동적으로 그 디렉토리를 액세스하게 된다.

예를 들면 다음과 같다.

```
SUBST E: C:\LOTUS\DATA\KIM
```

이렇게 설정해두면 DIR E: ↵에 의해 C:\LOTUS\DATA\KIM\의 내용이 출력되게 된다. SUBST만 입력하면 현재 설정된 가상 드라이브의 이름

과 설정 상황을 보여준다. SUBST E: / D는 위에서 지정한 가상 드라이브를 해제한다.

이때 가상 드라이브 문자는 아무 것이나 되는 것이 아니고 현재 사용하고 있는 컴퓨터의 하드웨어와 관계가 있다. 그 드라이브가 논리적으로 5개의 드라이브를 다룰 수 있다면 사용할 수 있는 문자는 A:~E:까지이나 16개의 드라이브를 다룰 수 있다면 A:~P:까지 사용 가능하다.

MORE.COM

[형식] ~ ¦MORE
여기서 ~는 DOS 명령어이다.

MORE.COM은 화면에 출력할 내용들을 한 화면 단위로 출력하는 기능을 한다. 화면에 출력될 내용이 많을 경우에 화면이 위로 스크롤되는데 MORE 코맨드를 이용하면 한 화면씩 볼 수 있다. 한 화면이 가득차면 화면의 맨 밑에 '---More---'라는 메시지를 출력하고 키 입력을 기다리게 된다. 이때 <Ctrl>+<C>를 제외하고 아무키나 누르면 다음 화면이 계속된다.

MORE는 긴 텍스트 화일을 한번에 한 화면씩 볼 때 유용한 명령이다.

```
TYPE READMO.DOC ¦MORE
DIR B: ¦MORE
```

이것은 TYPE 명령과 DIR 명령에 MORE를 이용한 예이다. MORE를 수행하기 위해서 DOS는 디스크에 작업영역을 만들고 한 화면분의 자료를 임시로 기억시켜두는 과정을 반복한다. 따라서 디스크에 전혀 여분이 없다면 MORE명령은 수행할 수 없게 된다. DIR명령의 경우에는 DIR / P가 한 화면씩 보여주는 것이고, 이들중 MORE를 이용하는 것이 더 많은 시간이 걸린다.

DISKCOMP.COM

[형식] DISKCOMP [D1:] [D2:] [/1] [/8]

이것은 2개의 디스켓의 내용을 비교하라는 내용이다. 즉 이 코맨드는 디스켓을 복사한 후 제대로 복사가 되었는지 등을 알기 위해 디스켓을 서로 비교하는 명령이다. '/1'은 한 면만을 비교하라는 것이고, '/8'은 그 디스켓이 1트랙당 8섹터로 포맷되어 있음을 의미한다.

디스켓을 비교하다 내용이 다른 부분이 있으면 'Compare error(s) on Track xx, side xx'와 같은 메시지가 출력되며, 비교후 이상이 없는 경우에는 'Diskettes compare OK'라는 메시지가 출력된다. DISKCOMP.COM은 DISKCOPY로 디스켓을 복사한 후 유효하며, COPY 코맨드를 이용하여 화일단위로 복사를 하게되면 화일의 순서가 일치하더라도 Diskcompare는 대부분 실패하게 된다.

또한 복사하는 디스켓과 복사를 받는 디스켓의 화일의 순서가 같다고 할지라도 그 화일이 저장된 물리적인 위치는 대부분 서로 다르게 된다. DOS는 화일을 기록할 때 디스켓의 비어있는 위치에서부터 기록한다.

만약 그 디스켓에서 삭제된 화일이 있다면 DOS는 그곳에 화일을 기록하고, 만약 삭제된 부분이 기록할 부분보다 작다면 기록하고 남는 부분은 다른 곳에 기록되게 된다. 그렇기 때문에 디렉토리상의 화일의 순서가 같더라도 그 화일이 기록된 위치는 다르게 되는 셈이다(그림 1 참조).

화일이 여러개의 공간에 분할되어 저장되면 그 화일을 읽어들이는 시간이 더 걸리게 된다. 이렇게 분할된 화일들은 빈 디스켓에 화일단위로 복사를 하면 연속된 영역에 저장된다. 그러므로 화일의 삭제가 빈번한 디스켓은 가끔씩 COPY 명령으로 복사해서 쓸 필요가 있다.

CHKDSK.COM

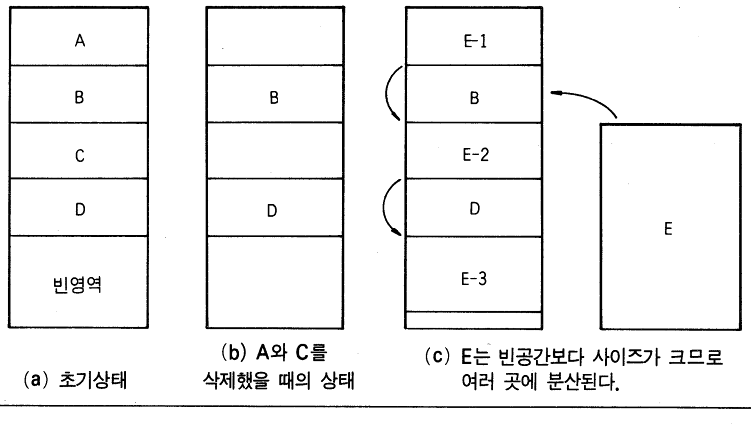
[형식] CHKDSK [D:] [/F] [/V]

이 코맨드는 디스켓상의 결함을 검사하는 것으로 '/F'는 디스크에 에러가 발생했을 때 자동적으로 보수해주며, '/V'를 설정하면 디스크의 화일을 표시해준다. 즉 이 코맨드는 디스크의 모든 화일을 읽어들여 이상이 있는지를 검사한다. 디스크상의 결함이란 여기서는 클러스터의 결함을 의미한다.

DOS는 화일을 클러스터(Cluster)라는 단위로 취급한다. 클러스터란 2개의 연속된 위치에 있는 2개의 섹터를 묶은 것인데, 데이터가 여러개의 클러스터에 저장된 경우에 다음 데이터가 어느 클러스터에 기록이 되어있는지를 DOS가 알고 있어야만 제대로 읽어들일 수 있게 된다.

따라서 DOS는 디스켓의 부트스트랩

〔그림 1〕 화일이 여러 부분에 분산되어 저장되는 과정



로더 다음 영역에 FAT(File Allocation Table)이라는 영역을 설정해두고 그곳에 클러스터의 연결상황을 표시해 둔다. DOS가 화일을 읽을 때에는 이곳을 참조하기 때문에 여러 클러스터에 분산되어 수록되어 있는 자료라 할지라도 제대로 읽어들이게 된다. 그렇기 때문에 FAT는 부트 스트랩 로더 다음으로 중요한 곳이다.

디스크를 사용하다 보면 전혀 예기치 않게 FAT가 자기적으로 혹은 물리적으로 손상을 입을 수가 있다. 이 경우에는 화일이 여러 조각으로 부서지기도 하고, 엉뚱한 화일이 서로 연결되기도 한다. CHKDSK 코맨드는 디스크의 FAT에 이상이 있는지의 여부를 조사하여, 이상이 생긴 경우에는 보수(/F)를 해주기도 한다. 즉 FAT에 이상이 생겨 화일이 여러 조각으로 분산된 경우에 CHKDSK는 'FILE0001. CHK'로 잘라진 조각들을 각각 독립된 화일로 만들어 준다. 이것은 나중에 에디터나 워드프로세서를 이용하여 서로 연결할 수 있다.

단 CHKDSK 코맨드에 의해 에러가 발견된 디스크는 모든 화일을 다른 곳으로 복사하여 사용하고 그 디스크는 다시 포맷하는 것이 좋다. 그리고 CHKDSK 코맨드는 서브 디렉토리는 체크하지 않으므로 서브디렉토리의 화일을 체크하는 경우라면 드라이브명 다음에 서브디렉토리를 지정하여야 한다.

'/F' 옵션으로 화일을 복구하는 경우 전혀 예기치 않는 결과를 초래할 수 있다. 그러므로 먼저 옵션없이 CHKDSK 코맨드를 사용하여 디스크를 체크한 다음 에러가 발생하면 먼저 그 디스크의 화일을 전부 백업하고나서 에러복구작업을 하는 것이 좋다.

이와 같이 CHKDSK 코맨드는 손상입은 화일을 일부나마 복구하려는 경우에 사용하는 최후의 수단이므로 CHKDSK 가 화일을 완전하게 원상태대로 복구해 주지는 못한다. 그러므로 귀중한 자료인 경우에는 항상 백업을 해두어 만일의 사태를 대비하는 것이 좋다.

SORT.COM

[형식] SORT [D:] [/R] [/+n] 〈화일명1〉 [〉화일명]

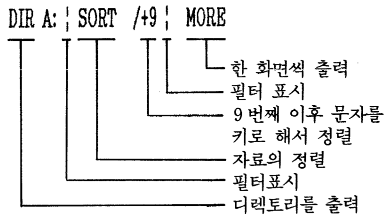
이 코맨드는 텍스트화일을 크기순으로 정열하는 것으로, '〉화일명'을 지정한 경우에는 정렬한 결과를 새로운 화일로 만들어 주지만 생략하면 결과를 화면에 표시한다. '/R'은 내림차순(큰 것 부터 작은 것 순)으로 정렬하며 '+n'은 n번째 이후의 문자열의 크기를 비교한다.

그러나 SORT 코맨드로 정렬하는 경우는 텍스트화일에 한하기 때문에 바리너리 화일을 소트하게 되면 그 화일은 DOS상에서 실행시킬 수 없게 된다. 텍스트 화일이란 에디터나 워드프로세서 등에서 작성한 것으로 매 라인의 끝에 리턴 코드(16진수 0D 0A)를 가지고 있는 것을 말한다.

SORT와 MORE를 DOS에서는 필터(Filter)라고 부르는데 필터는 앞의 코맨드의 처리결과를 다시 처리하는 명령이라고 생각하면 된다. 필터명령은 그 명령 앞에 '¦'를 붙여서 사용한다. 화일 디렉토리를 확장자의 순으로 정렬하여 화면단위로 출력하는 것을 예로 들면 다음과 같다.

```
DIR A:¦SORT/+9¦MORE
```

이것은 다음과 같은 의미이다.



PRINT.COM

PRINT.COM은 DOS의 명령어를 수행하면서 텍스트 화일을 프린터에 출력해주는 프린터 스풀러이다.

스풀러는 컴퓨터의 시간을 쪼개서 간간히 프린터를 지원하여 컴퓨터와 프린터를 동시에 사용하게 해주는 소프트웨어나 하드웨어를 말한다. 컴퓨터쪽에서 보면 프린터와 키보드 혹은 프로그램을 교대로 수행하는 것이겠지만 사용자의 입장에서 보면 프린터와 컴퓨터가 별개로 운용되는 것처럼 보인다. 즉 프린터는 프린터대로 출력을 하면서 우리는 프린터와는 전혀 별개인 프로그램을 수행할 수 있다.

PRINT.COM을 사용하여 출력할 수 있는 화일은 표준 아스키 화일이어야 한다. 이를테면 보석글 등의 워드프로세서 등에서 작성된 화일은 확대, 축소, 문자 등이 출력되지 않을 수도 있다. 그 이유는 워드프로세서마다 확대, 축소 등의 특수한 형태의 문자체를 지정하는 방법이 다르기 때문이다. DOS에서는 경우에 따라 이들 특수 문자를 다르게 인식하기도 한다.

[형식] PRINT [D:] 〈화일명〉〈화일명〉..../옵션

화일명을 공백으로 분리하여 여러개를 기술하면 DOS는 그 화일들을 차례대로 프린터에 출력한다. 화일명은 큐(Que)라고 하는 임시 기억장소에 저장된다.

[옵션]

/T : 현재 큐에 등록되어 있는 모든 화일을 없앤다.
/C : 지정된 화일을 큐에서 삭제한다. /C는 화일명 다음에 붙인다.
/P : 지정된 화일을 큐에 더한다. /P는 화일명 다음에 붙인다. 여러개의 화일을 더하는 경우에는 처음의 화일에만 붙여도 된다.
/Q:n : 큐의 크기를 설정한다. n으로 지정할 수 있는 값은 4~32이며 생략시는 10이다.

(예) PRINT B:LIST-1/C B:LIST-2/P B:LIST-3 B:LIST-4

큐에서 LIST-1을 제거하고 B:의 LIST-2, LIST-3, LIST-4를 큐에 더하라는 내용이다.

(예) PRINT /T

큐에서 모든 화일을 제거하는 뜻이다.

RECOVER.COM

[형식] RECOVER [D:] [화일명]

이 코맨드는 디스크에 불량 섹터가 있을 경우 그 부분을 도려내어 디스크를 사용 가능한 상태로 만든다.

RECOVER는 화일명을 생략하면 디스크 전체를 검사한다. 이때 발견되는 화일은 'FILE0001.REC', 'FILE0002.REC'… 등으로 이름이 변경되어 루트 디렉토리에 수록된다. 그러므로 RECOVER로 디스크를 복구한 후에는 REN 코맨드로 다시 원래의 이름으로 변경을 하여야 한다.

RECOVER로 복구되어 'FILExxxx.REC' 등으로 이름이 바뀐 화일은 원래의 이름과 순서가 일치하므로 RECOVER 를 행하기 전에 먼저 화일 목록을 프린터 등으로 출력하고 이 코맨드를 수행할 필요가 있다. 그렇지 않았다면 이 화일들을 원래의 이름으로 되돌리는 것은 불가능할 것이다.

이런 경우를 대비하는 제일좋은 방법은 데이터를 수기적으로 백업해두는 일이며, 만일 백업하지 않은 상황에서 에러를 복구해야 하는 경우에는 DISKCOPY 코맨드로 에러가 생긴 디스크를 일단 복사하고, 이 복사본 디스크를 가지고 작업을 하는 것이 좋다. 왜냐하면 빈대 한 마리를 잡겠다고 초가 삼간을 태운다는 말이 있듯이 에러가 생긴 화일하나를 복구하느라고 디스크 전체를 망가뜨릴 수가 있기 때문이다.

RECOVER는 화일을 검색하여 그 화일이 불량 섹터를 포함하고 있다면 그 부분을 도려내어 데이터를 수록하지 않도록 한다. 그러므로 그 화일이 데이터 화일인 경우에는 일부가 잘려지게 되고, 프로그램인 경우에는 복구가 되어도 실제로는 사용이 불가능하게 된다. 따라서 RECOVER로 화일을 복구하는 경우는 최후에 시도해보는 마지막 카드인 만큼 실행하기에 앞서 충분한 주의가 필요하다.

TREE.COM

[형식] TREE [D:] [/F]

이것은 디스크의 서브디렉토리의 구조(트리 구조)를 화면에 표시한다. '/F'를 붙이면 그 디렉토리에 있는 화일명도 함께 표시한다. 이 코맨드는 서브디렉토리가 많이 설정되어 있는 하드디스크의 구조를 표시할 때 유용한 명령이다.

단지 아쉬운 것은 이 코맨드가 트리구조를 그림으로 표시해주지 못한다는 점인데 출력결과를 '>PRN'을 붙여 프린터로 출력하면 서브디렉토리의 구조를 일목요연하게 볼 수 있다.

VDISK.SYS

[형식] DEVICE=VDISK.SYS [크기] [섹터당 바이트 수] [디렉토리의 크기][/E]

이것은 CONFIG.SYS에 설정되며, 램드라이브를 설정한다. 램드라이브는 메모리(RAM)의 일부분을 떼어내어 드라이브처럼 운용하는 것을 말한다. RAM 드라이브는 2개의 플로피 드라이브를 가지고 있는 기종에서는 C:를 배정 받으며, 하드 드라이브를 가지고 있는 기종에서는 D:나 E:를 배정받는다.

램드라이브는 순수하게 전자적으로 동작하므로 기계적으로 동작하는 플로피 드라이브에 비해 200배 정도 고속이다. 특히 게임등의 프로그램을 실행하는 경우에는 디스크의 모든 화일을 RAM 드라이브로 복사하고 RAM 드라이브에서 실행을 시키게 되면 화면처리가 대단히 빨라지게 된다. 램드라이브는 전원이 꺼지면 자동적으로 소멸되므로 중요한 자료인 경우에는 플로피 디스크나 하드 디스크로 자주 백업을 하도록 하여야 한다.

'크기'는 RAM 드라이브의 크기를 설정하는 것인데 생략시는 64KB로 지정되는데, 보통 128, 256, 384, 512 등으로 지정할 수 있다. '섹터당 바이트 수'는 128, 256, 512 등으로 지정할 수 있는데 생략시는 128이며, 플로피 디스크의 경우는 섹터당 256바이트이다. '디렉토리의 크기'는 램드라이브에 수록할 수 있는 최대 화일수를 설정하는데 있어서 생략시는 64로 된다.

'/E'는 확장 메모리카드를 장착한 경우에 확장된 메모리를 사용하도록 하는 스위치이다. DOS에서는 최대 640KB 의 메모리를 가질 수 있으며, 그 이상의 메모리는 DOS가 직접 액세스할 수는 없고 램드라이브 등으로 사용하게 된다.

램드라이브의 용량을 너무 크게 잡으면 사용자 메모리가 작아지므로 응용프로그램이 수행되지 않는 경우도 종종있다. 예전에는 256KB이면 어지간한 소프트웨어는 다 수행 되었으나 요즈음은 최소한 384KB이상이 되어야 수행되는 것들이 많다. 그러므로 응용프로그램을 수행하는 경우라면 128KB 정도이어야 하고, 게임을 실행할 경우라면 384KB로 설정해 두는 것이 적당하다.

배치화일에서 파라미터의 전달

앞에서 설명한 PRINT.COM의 경우 PRINT〈화일명〉으로 PRINT.COM을 기동한다. PRINT.COM은 확장자가 .COM이므로 PRINT만을 입력하면 DOS에서 실행할 수 있으나 〈화일명〉을 기술하면 그 화일명을 PRINT.COM이 전달받는다. 이와 같이 프로그램 등에 전달하는 일련의 문자열을 파라미터(Parameter)라고 부른다.

배치화일은 실행중 파라미터를 전달받는 기능이 있다. 파라미터는 %0~%9로 10개까지 가능하다.

우선 간단한 예로 베이직을 실행하는 경우를 생각해보자. GWBASIC〈화일명〉↵에 의해서 DOS는 베이직을 실행하며, 베이직은 다시 지정한 화일을 로드한 다음 자동으로 실행하게 된다. 이때의 화일명을 파라미터로 주면 베이직의 실행시 지정한 프로그램을 실행시킬 수 있게 된다.

```
A> COPY CON BS.BAT
CLS
GWBASIC %1
<Ctrl-Z>
```

일단은 이와 같은 배치화일을 작성해 보자. 여기에서 '%1'은 파라미터를 나타낸다. 이 프로그램을 실행하는 경우에는 BS TEST ↵를 입력하면 'TEST' 가 '%1'에 대입되어 전달된다. 컴퓨터는 GWBASIC을 실행하게 되고, GWBASIC은 '%1'에 기억된 프로그램(여기서는 'TEST')을 로드하여 실행하게 된다.

DOS에서는 %0~%9를 특별히 환경변수라고 부르기도 한다. 이런 환경변수는 고급언어의 컴파일시나 어셈블시에 아주 효과적이다. 어셈블리 언어로 짜여진 프로그램을 어셈블하는 경우라면 다음과 같이 작성할 수 있다.

```
A> COPY CON ASM.BAT
MASM %1;
LINK %1;
<Ctrl-Z>
```

어셈블을 하는 경우라면 ASM〈화일명〉에 의해 어셈블과 링크를 자동적으로 수행하게 된다.

FOR (내부 명령어)

FOR 코맨드는 TYPE문처럼 와일드카드 문자('*', '?')를 사용하지 못한 명령어를 처리할 경우 일괄적으로 모아서 처리하는데 이용된다. 이것은 배치처리에도 이용할 수 있고, 키보드에서 직접 입력하여 처리할 수도 있다.

```
[형식]  FOR %변수 IN(요소,요소,...) DO
          〈코맨드〉 %변수      ← 키보드 입력
        FOR %%변수 IN(요소,요소,...) DO
          〈코맨드〉 %%변수     ← 배치화일
```

여기서 〈변수〉는 0~9가 아닌 임의의 문자이다. 〈변수〉에 수치를 쓰면 DOS는 이것을 파라미터로 인식하게 된다. 이를테면 KIM.DAT, LEE.DAT, PARK.DAT 등의 3개의 화일을 TYPE문으로 모두 화면에 표시하는 것을 FOR문을 사용해서 처리하면 다음과 같다.

```
A>FOR %A IN (KIM.DAT LEE.DAT PARK.DAT) DO
                              TYPE %A
```

이 경우에는 확장자가 모두 .DAT이므로 다음과 같이 입력할 수 있다.

```
A>FOR %A IN (KIM LEE PARK) DO TYPE %A.DAT
          └─────────────────────────┘
           변수명('A')이 서로 대응한다.
```

이것을 배치화일에서 사용하는 경우라면 변수명 앞에 '%'를 2개 기술해야 한다. 예를 들어 화일명을 여러개 입력하여 그 화일의 내용을 화면에 출력하는 배치화일을 작성하면 다음과 같다.

```
A> COPY CON TP.BAT
FOR %%X IN (%1 %2 %3 %4 %5) DO TYPE %%X
<Ctrl-Z>
```

여기서는 화일명를 최대 5개까지 입력받을 수 있게 했는데 더 많은 화일명을 입력받고자 할 경우라면 '%6', '%7'… 등으로 계속 기술하면 된다. 위의 '%1', '%2'… 등은 앞에서 배운 파라미터이다.

A>TP〈화일명〉〈화일명〉… ↵에 의해 DOS는 그 화일들의 내용을 화면에 모두 표시(TYPE)해 줄 것이다. 또 화일이 KIM-1.DAT, KIM-2.DAT, KIM- 3.DAT… 와 같이 기록되어 있는 경우라면 위의 DO 다음을 이렇게 기술하면 된다.

```
FOR %%X IN (%1 %2 %3 %4 %5)
        DO TYPE KIM-%%X.DAT
```

이때는 TP 1 2 3 4 ↵에 의해서 KIM-1.DAT, KIM-2.DAT, KIM-3.DAT, KIM-4.DAT가 모두 화면에 표시될 것이다.

또 한가지 예로 확장자가 .BAK, .OBJ, .EXE인 화일을 모두 삭제하는 것을 FOR문으로 입력하면 다음과 같다.

```
FOR %A IN (BAK OBJ EXE) DO DEL *.%A
```

끝으로

필자는 직업이 컴퓨터 학원장이기 때문에 하루 일과를 컴퓨터 스위치를 켜는 것으로 시작해서 컴퓨터를 끄는 것으로 마치게 된다. 이 강좌는 지난 수년간 초보자를 대상으로 실시하는 본 학원의 MS-DOS 강좌를 3회에 결쳐 간략하게 소개한 것이다.

지면상의 이유나 표현상의 문제 등으로 인해 더 많은 정보를 전달하지 못했음을 아쉽게 생각하며 이 강좌가 다소나마 독자들의 컴퓨터 생활에 도움이 되었기를 바라마지 않는다.

이 강좌는 컴퓨터를 처음으로 대하는 사람들을 대상으로 시작한 만큼 DOS의 특수한 처리 등은 기술하지 않았으므로, 이 강좌를 다 이해한 독자들은 좀 더 전문서적을 참고하기 바란다.

알림

3회에 걸쳐 독자여러분의 성원속에 연재된 'MS-DOS 입문' 강좌는 이번회를 마지막으로 끝마치게 되었읍니다.
그동안 집필해주신 박승제님께 감사드리며, 다음부터는 다른 내용으로 본란을 꾸미고자 하오니 계속해서 많은 성원을 보내주시길 바랍니다.

둘리가 본 컴퓨터세계 3

컴퓨터와 선보는 요령

이만희 /한국교육개발원 컴퓨터교육 연구센터

그대 있음에 내가 있네

(1) 투자와 가치

학교에 다니는 자녀가 있다면 학교에서 컴퓨터를 배운다든가 옆집 아이가 컴퓨터학원을 다닌다는 이야기를 들었을 것이다. "맹모삼천지교"라는 말도 있는데 부모라면 당연히 "우리집 아이가 놀기만 하다가 컴퓨터시대에 적응하지 못하고 뒤떨어지면 어쩌나" 하는 마음으로 컴퓨터 판매점에서 한번쯤은 유심히 컴퓨터와 가격표를 살펴보기 마련이다.

또 대학생이라면 다른 친구가 컴퓨터로 산뜻하게 인쇄한 레포트를 제출하는 것을 보았을 것이다. 천재는 졸필이라고 하지만 삐뚤 삐뚤하게 쓴 자신의 글씨를 스스로 보아도 한심한 느낌이 드는 사람이 많다. 더우기 냉정한 교수님이 자신의 레포트를 읽으시며 지으실 표정을 생각하면 부모님을 설득해서라도 꼭 컴퓨터를 마련하고자 한다.

(2) 정보와 돈

밀실에서 조용히 흘러나온 도시계획이나 고속도로 계획에 대한 정보는 곧 큰 돈으로 될 수 있다. 이렇게 비싼 정보는 아니지만 서울의 어느 슈퍼마켓에서 팔리는 배추가격 역시 돈이 되는 정보이다.

슈퍼에서 한포기에 천원이나 천오백원에 팔리는 배추도 두메산골 배추밭에서는 백원, 이백원한다.

만일 그 농부가 슈퍼와 직접 연락하여 가격과 소요량에 대한 정보를 교환할 수 있고, 최소 비용의 수송방법에 대한 정보만 가질 수 있다면 결코 헐값에 배추를 넘기진 않을 것이다.

그런 꿈같은 정보수단이 아니라도 서울 시내와 같이 20원의 전화요금으로 서울에 있는 슈퍼와 통화를 해서 정보를 교환할 수만 있어도 그 농부의 수입은 당장 몇배로 늘어날지 모른다. 불행하게도 현재는 가난한 농부가 비싼 시외 통화요금을 부담해야 하고, 대도시 여러 곳에 전화를 걸기에는 배보다 배꼽이 커진다. 즉, 그 농부는 정보로부터 소외될 수 밖에 없고 정보의 소유자에게 배추를 헐값으로 넘겨야만 한다.

농부와 배추가 이렇다면 국제간의 무역전쟁은 말할 필요도 없을 것이다. 대규모의 데이터베이스와 국제적인 컴퓨터 네트워크를 소유한 선진국은 우주에 떠있는 인공위성으로 세계 모든 지역의 농산물 작황을 감시하며 TV도 통신위성에서 직접 송출하여 정보의 사각지대를 없애고자 하고 있다.

미국의 농부가 컴퓨터 네트워크로 전국의 곡물시세를 파악하고 최소비용의 수송수단을 찾아낸 후 판매계약을 하듯이 우리도 그러한 사회로 가려면 국가적으로 컴퓨터의 보급과 네트워크의 구축을 서둘러야 한다.

그러나 국가의 혜택만을 마냥 기다리다 보면 언제쯤 자기 차례가 올지 모른다.

역시 많은 부담이 되지만 스스로 구입하는 편이 빠르고 확실할 것이다.

중매장이의 허풍

컴퓨터를 살 때 가장 흔한 방법은 주위에서 잘아는 컴퓨터 전문가에게 물어 보는 것이다. 그러나 전문가의 말을 들어도 쉽게 이해하기 어렵고 더욱 고르기가 어려워진다. 중매장이 중에는 별 볼일없는 사람도 좋은 점만을 침소봉대하여 말하는 경우가 있다. 그러나 그것이 전혀 근거없는 말은 아니고 단지 앞으로 그렇게 될 장래성이 보이는 것을 좀 부풀려 말하는 것이다. 그런 희망도 없다면 누가 결혼을 하겠는가.

컴퓨터도 구입하기 전에 앞으로 변해갈 미래를 한번 살펴 보는 것이 좋다. 물론 여기서 소개하는 것들 모두가 모든 문제를 당장에 해결할 수 있는 컴퓨터는 거의 없다. 단지 점차 발전해 가고 있다는 것이다. 컴퓨터를 사고자 하는 이유는 당장의 편리함을 위해서만이 아니고 자신의 장래에 대비한 투자도 있다. 따라서 지금 당장은 아니하더라도 다음에 더 갖추어야 될 것이 무엇인가는 알아야 한다.

(1) 환상의 오디오와 컴퓨터

음악에 관심이 있는 분이라면 콤팩트디스크(CD)의 소리를 들어 보았을 것이다. 피아니스트의 옷자락스치는 소리와 숨소리마저 들려오는 생생한 음질에 반하는 이도 있고 너무 현대적인 느낌에 질려 버리는 사람도 있다. 오색영롱하게 반짝이는 작은 은빛 원반에 아름다운 소리가 기록된다는 사

실이 신비한 느낌을 주어 "환상의 오디오"라고도 부른다.

이 콤팩트디스크는 음악대신 컴퓨터의 기록을 저장하는데 사용할 수도 있다. 뉴스에 자주 소개되는 램(RAM)은 컴퓨터의 중요한 기억장치이지만 전기를 꺼버리면 그안에 들어 있는 자료가 모두 없어지기 때문에 미리 디스크에 보존해야 한다. 그러나 보통의 컴퓨터 디스크는 보통 책 한권정도 기록할 정도의 용량으로 백과사전과 같은 방대한 자료의 영구적 보관에는 문제가 있다. 그런데 요즘 오디오 분야에서 각광을 받고 있는 콤팩트디스크로 이 문제를 간단히 해결할 수 있게 되었다.

콤팩트디스크는 본래 원음을 보다 가깝게 재생하고자 컴퓨터와 레이저기술을 이용한 것으로 "광디스크" 또는 "레이저디스크"라고도 부른다. 일본 소니와 네덜란드 필립스사의 공동 개발로 발명된후에 미국, 일본, 유럽등 선진국에서 빠른 속도로 보급되고 있다(사진 1 참조).

① 컴퓨터와 오디오 겸용의 디스크 플레이어

콤팩트디스크는 훌륭한 음질 뿐 아니라 사용이 아주 편리하여 선진국의 경우 디스크 시장의 반 이상을 차지하고 있다. 디스크를 움직이는 플레이어도 대량생산과 경쟁으로 100만원에서 15만원 정도의 가격으로 낮아졌다. 디스크플레이어 내부는 마이크로프로세서가 들어 있어 자동적으로 정밀하게 제어할 수 있도록 되어 있다.

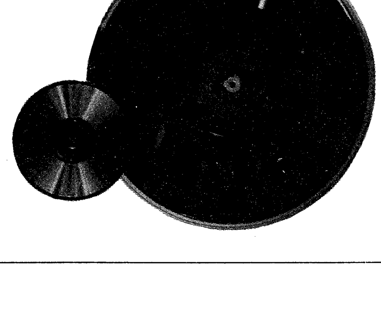
기록 원리도 일단 음의 크기와 형태를 숫자로 바꾸어 기록하고 음을 재생할 때는 기억된 숫자를 다시 계산하여 소리로 바꾸는 방식으로 컴퓨터가 디스크에 기록하는 것과 차이가 없다. 그리고 이미 몇몇 회사에서 플레이어 하나로 오디오와 컴퓨터 양쪽에 연결하여 사용할 수 있는 겸용의 디스크플레이어도 만들고 있다.

② 컴퓨터에서 콤팩트디스크의 활용

콤팩트디스크(CD-ROM)는 직경 12㎝의 은색 원판으로 손바닥 안에 들어가는 정도이지만 550 메가바이트의 엄청난 용량을 저장할 수 있다. 이 용량은 360 킬로바이트인 보통의 컴퓨터 플로피디스크의 1,500배 이상이다. 즉 천오백권의 책을 콤팩트디스크 한장에 담을 수 있다는 말이 된다. 퍼스널컴퓨터에서 이 콤팩트디스크를 이용하게 되자 엄청난 기억능력으로 새로운 "환상의 기억장치"로 등장하고 있다. 유명한 도서관이나 연구소 등에서만 소장하던 수많은 책과 자료도 콤팩트디스크에 기록하면, 불과 몇장의 디스크안에 담아 가정집에 보관하고 편리하게 이용할 수 있게 된다.

그 예로서 데이터베이스의 콤팩트디스크화이다. 데이터베이스 회사는 어느 특정 분야의 정보를 전부 모아서 컴퓨터에 입력해 놓고 더 체계화된 정보를 상품으로 판매하는 회사이다. 데이터베이스는 자연히 많은 정보가 쌓여 종래의 디스크로 저장하기에는 부피가 너무나 크고 비용이 많이 든다. 따라서 데이터베이스 회사가 대형 컴퓨터를 운영하여 여기에 연결된 여러 사용자의 이용시간을 계산해 비용을 부과한다. 의학분야의 데이터베이스 회사라면, 새로운 병의 치료법 발견이나 새로운 약의 발명에 대한 정보를 모아서 입력하고, 여러 병원에서는 돈을 내고 컴퓨터를 통해 입력된 정보를 이용하는 것이다. 그러나 사용시간에 비례하여 요금을 내야 되기 때문에 느긋한 마음으로 정보를 이용하기가 어렵다.

〈사진 1〉 콤팩트디스크



콤팩트디스크에 데이터베이스를 담으면 대량 생산될 경우 불과 몇만원 정도의 비용으로 개인이 필요한 정보를 구입할 수 있게 된다. 한장의 콤팩트디스크에 기억되는 정보는 1초에 30자씩 읽을 때 하루 24시간을 계속 읽어도 한달 이상의 시간이 걸리는 막대한 양이다. 따라서, 커다란 백과사전이나 잡지, 두꺼운 전화번호부를 콤팩트디스크로 만든다면 단 한장에 기록되고 비용이 적게 든다. 더우기 컴퓨터를 사용해 디스크에서 필요한 부분을 쉽게 찾을 수 있으며 손으로 다시 입력하지 않고도 컴퓨터의 자료로 즉시 이용할 수 있기 때문에 워드프로세서로 글을 쓰는 사람이나 법률가, 교사 등 폭넓은 분야의 사람들이 이용할 수 있다. 또한 콤팩트디스크는 문자 정보 뿐 아니라 소리나 천연색 사진까지 저장할 수 있다.

예를 들어 세계 각국의 개구리에 대한 자료를 데이터베이스화 해서 콤팩트 디스크에 저장하면 개구리 종류마다 올챙이적부터의 사진과 자세한 설명, 그리고 개구리의 울음소리까지도 들을 수 있다. 이렇게 되면 책의 형태도 큰 변화가 올 것이다. 레코드처럼 생긴 광디스크 책이나 잡지를 파는 서점도 나올 것이다.

③ 광 디스크의 종류

레이저를 이용하는 광디스크도 단점은 있다. 아직은 컴퓨터의 디스크처럼 마음대로 기록하고 다시 지울 수 있는 기술이 충분하지 못하기 때문이다. 한번 기록되면 사진과 같이 쉽게 변하지 않는 특성을 갖는 광 디스크의 종류는 세가지로 사용분야가 각기 다르다.

● CD-ROM

녹음 테이프는 사용자가 마음대로 녹음할 수 있지만 실제로 자기 목소리를 녹음해서 감상하는 사람은 별로 없다. 대부분 유명 가수의 노래나 전문가들이 연주한 것을 구입해 즐겨 듣는다.

이와 마찬가지로 컴퓨터 자료도 보통 사람이 직접 작성하기 보다는 전문분야의 사람이 자료를 만들면 일반인은 그것을 사서 사용하면 된다. 또 대량 복제되는 자료는 그 질과 양에 비해 값이 아주 싸다.

유력한 신문같은 경우 대부분의 도서관이 몇년치씩을 보관 하느라 많은 공간을 필요로 하는데 이를 CD로 만들면 간단하게 대량 생산하여 값싸게 판매할 수 있고 공간도 거의 필요없다. 이렇게 내용을 바꾸지 않고 재생 전용 매체인 경우 컴퓨터의 ROM과 같이 내용을 바꿀 수 없어 CD-ROM이라 한다.

● **WORM(Write-Once Read-Many : 한번 기록 여러번 읽기)**

레이저디스크에 여러번 다시 쓰고 지우기는 어렵지만 깨끗한 판위에 한번만 기록하는 기술은 비교적 쉽다. 따라서 사무실의 문서를 축소 보관하기 위해 사용하는 마이크로 필름을 레이저디스크로 대치할 수 있다. 한번 기록하면 다시 지워지지 않는 것이 불편하지만 오히려 지워지면 안되는 기록을 위해서는 장점이 될 수도 있다.

● **삭제가능 광디스크(Erasable Optical Storage)**

아직도 연구 단계이지만 현재의 자기디스크보다 몇십배의 용량을 갖고 자유로이 쓰고 지울 수 있는 실험적인 광디스크도 나오고 있다.

(2) 인공지능

컴퓨터는 계속해서 보다 빠른 계산 속도와 많은 기억용량을 처리할 수 있도록 변해왔다. 그러나 아직도 인간의 두뇌에 비하면 능력이 훨씬 떨어지는 것이다. 심지어 조그만 곤충의 행동도 컴퓨터보다 훨씬 높은 지능을 나타낸다. 그동안 지능의 본질을 분석하고 컴퓨터로 이를 실현하려는 인공지능에 대한 연구가 진행되어 왔다.

인공지능을 이용한 것으로 특정 전문분야에서 조언을 해주거나 숙련도가 필요한 복잡한 작업을 수행하는 전문가 시스템을 들 수 있다. 전문가 시스템은 전문분야에 대한 경험과 여러가지 사실들을 조직적으로 처리하여 어떤 문제에 대해 검토 범위를 점차 좁혀 결론에 도달하는 것이다. 이 시스템은 의학분야에서 가장 먼저 사용되어 신체내부 장기분석이나 신호분석같은 분야에는 상당한 발전이 이루어졌다. 의사를 만날 수 없는 지역에서 의료전문가시스템은 질병의 상태를 분석해서 병의 종류를 알아내고 약이나 치료법에 대한 자료를 알려주는 등 만나기 어려운 전문가의 지식을 대신해 준다.

(3) 음성인식 및 합성

컴퓨터에 자료를 입력할 때 아직까지는 키보드를 주로 이용하는데, 필자와 같이 타자에 익숙하지 않은 사람에게는 정말 불편하다. 그러나 이제는 우리가 말하는 대로 받아 적는 비서처럼 사람의 음성을 알아듣는 컴퓨터가 나오고 있다.

음성으로 입력하면 타자기의 4배, 필기의 10배 속도로 입력이 가능하다. 음성인식을 위해서는 컴퓨터 시스템을 사람의 목소리에 반응하도록 훈련시켜야 한다. 개인에 따라 음성이나 발음방식에 차이가 심하기 때문에 모든 사람의 음성을 알아 듣기에는 아직 문제가 있다. 또 외부 잡음의 배제, 연속 음성의 구별, 문장 속의 단어식별 등 어려운 문제가 있다.

그래도 간단한 숫자 몇개 정도는 여러 사람의 목소리를 구별할 수 있는 것도 있다. 일본에서는 은행의 잔고를 알기 위해 손님이 전화를 걸면 음성인식 컴퓨터가 전화를 받아 손님의 저금통장 번호를 알아 듣고 필요한 자료를 컴퓨터 음성합성을 통해 알려 주고 있다. 음성인식 단어가 단순한 숫자 정도이지만, 여러 사람의 목소리를 대부분 알아 듣는다는 것은 놀라운 일이다.

반대로 사람 목소리를 만들어 내는 음성합성은 입력된 단어를 음성패턴이 기록된 컴퓨터내의 사전에서 찾아낸다. 간단한 인사말이나 숫자라면 쉽게 만들어 낼 수 있어 이미 큰 병원 같은 곳에서는 음성합성을 이용하여 손님의 번호를 부르는데 이용하고 있다.

(4) 언어번역

이미 우리들이 일상생활에서 쓰고 있는 말을 컴퓨터를 사용하여 번역하는 것이 상품으로 판매되고 있다. 언어의 번역에서는 자연어의 의미 등을 정확하게 번역하기 위한 데이터베이스를 갖추고 있어야 한다. 언어번역에 있어서도 1종 언어번역(예 : 영어와 불어, 한국어와 일어), 2종 언어번역(예 : 중국어와 영어), 3종 언어번역(예 : 한국어와 영어) 등 언어의 멀고 가까움에 따른 어려움의 정도가 다르다.

현재 개발된 대형 컴퓨터를 사용한 시스템의 경우, 음성인식장치를 사용해 입력이 가능하고 시간당 일어와 영어를 6000 단어 정도의 속도로 번역하고 있으나 최종 번역물을 위해서는 사람이 다시 한번 다듬어야 한다. 이것은 아직 사람의 두뇌속에 들어 있는 수많은 일반상식을 컴퓨터가 데이터베이스로 갖추고 있지 못하기 때문이다. 그러나 객관성의 유지 등에 있어 사람을 통한 번역 보다 공정을 기할 수 있다. 퍼스널컴퓨터를 이용한 언어번역 시스템도 개발이 되어 일어, 영어 번역의 경우 큰 인기를 얻고 있으며, 또한 일어와 한국어 번역시스템의 개발도 완료되었다(그림 1 참조).

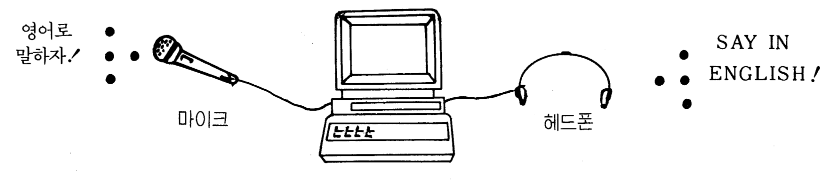
현재 사용되고 있는 4세대 컴퓨터보다 한단계 높은 차원의 성능을 갖도록 개발되고 있는 제5세대 컴퓨터가 실현되면, 인공지능을 활용해 사람이 번역을 하는 것처럼 유연성있게 상식에 맞는 번역을 할 수 있게 될 것이다.

컴퓨터의 결혼조건

(1) 가격

컴퓨터의 가격에는 여러 계층이 있

〔그림 1〕 언어번역시스템



지만 비싼 가격이 반드시 좋은 성능을 대별해 주는 아니다. 오히려 비싼 가격은 컴퓨터회사의 유명도와 비례한다고 할 수 있다. 유명한 컴퓨터메이커는 사용자들이 컴퓨터를 보다 쉽게 사용하도록 새로운 기술 개발에 많은 돈을 투자한다. 큰 돈을 들여 좋은 기능의 새로운 컴퓨터나 새로운 컴퓨터 프로그램을 개발하면 다른 회사는 몇달 후 곧 그것을 쉽게 모방해서 더 싼 값으로 판다. 그래서 앞서 가는 회사는 유명해 지지만 돈이 많이 든다.

또 유명회사가 그 명성을 유지하려면 이미 자사의 컴퓨터를 구입한 많은 고객들에게 친절한 서비스를 계속해야 한다. 그 회사는 여러 명의 경험 많은 전화 상담자를 고용해서 고객이 전화로 답답한 질문을 아무리 지루하게 계속해도 해결방안을 친절하게 알려주어야 하고, 때로는 무책임한 고객이 컴퓨터가 견디기 어려운 가혹한 환경에서 사용하다 고장을 내도 즉시 달려가 고쳐 주고 기계에 쌓인 먼지 청소까지 해주는 고급 기술자들을 유지하기 위해 많은 월급을 줘야 한다. 이런 기술개발이나 서비스는 컴퓨터 구입시 모든 컴퓨터 회사가 보장한다고 말하지만 성의만 갖고는 되지 않는다. 그들도 역시 돈이 필요하다 보니 컴퓨터 판매가격에 이런 비용을 포함시키다 보면 자연히 비쌀 수밖에 없다.

남보다 싼 가격으로 컴퓨터를 구입하면서 남보다 더 좋은 서비스를 바란다면 지킬 수 없는 약속을 강요하는 것과 같다. 약속이 지켜지지 않으면 사용자 스스로 해결하는 방법 밖에 없으므로 그때 소비하는 시간을 고려해서 알맞는 가격의 회사를 결정해야 한다.

이것은 여러 수준의 식당에서 식사하는 것과 유사하다. 비싼 식당이 반드시 맛있는 음식을 내오는 것은 아니지만, 분위기와 종업원들의 서비스는 음식가격과 비례하는 것이 보통이다.

최고급 의상실에서 컴퓨터로 단골고객의 사진이나 체형, 취향, 생일 심지어는 음성까지 입력해 손님의 기분을 가장 잘 맞추어 주는데 사용하려고 할 때 어려운 컴퓨터 사용법을 혼자 익히고 개발하며 고장시에 수리하느라 시간을 허비하는 것보다 최고급 컴퓨터를 구입해 컴퓨터 전문가의 안정되고 신속한 지원을 받으면서 옷을 하나 더 파는 편이 이익이 더 클수도 있다. 또는 신속히 고객에게 봉사해야 하는 회사같은 곳에서 컴퓨터가 고장나서 일이 늦어진다면 고객이 무척 화를 내고 다른 곳으로 옮겨 버릴 것이다.

반대로 어려운 형편의 학생이라면 컴퓨터가 고장나더라도 버스를 타고 판매점까지 직접 들고 갈 시간이 충분할테니 친구나 안면있는 사람이 소개하는 컴퓨터회사(소규모)에서 싼 가격의 컴퓨터를 선택하는 것이 바람직하다. 고객이 고장난 컴퓨터를 들고 올 때 외면하는 형편없는 컴퓨터 판매점은 거의 없다. 따라서 단순히 기능면에서는 같은 컴퓨터도 몇 배의 가격 차이가 있는 것이다. 만일 유명하지도 않고 가격만 비싸다면 특별한 이유가 있거나 바가지를 쓰는 경우이다.

(2) 보급율

인기있는 상품은 다 그만한 이유가 있다. 컴퓨터는 쉽게 살만큼 가격이 싼 물건이 아니므로 다른 사람도 컴퓨터를 살 때 여러가지를 알아 보고 비교하고 며칠씩 고민한 후에 기종을 결정했을 것이다. 따라서 여러 사람이 많이 산 컴퓨터라면 그만한 장점이 있다.

또 한가지 중요한 것은 다른 사람이 많이 갖고 있는 컴퓨터를 고르면 그들이 사용하는 소프트웨어를 쉽게 이용할 수 있고 프린터 등 값비싼 주변장치도 자신의 것을 구입하기 전까지는 잠시 신세를 질 수도 있다. 어느 특정한 회사의 컴퓨터가 유명해지면, 소프트웨어를 만드는 사람도 많이 팔린 기종에 맞추어 개발해야 판매시장이 넓어진다.

그러면 다른 하드웨어 회사에서는 그것과 동일한 규격의 컴퓨터를 만들어 보다 싸게 파는 것이 보통이다. 이렇게 하여 같은 규격의 컴퓨터가 거의 전체시장을 차지하게 되고 더 많은 컴퓨터 구입자들이 이런 컴퓨터를 선택하면서 컴퓨터 기종의 독점 현상은 더욱 심해진다. 16비트 컴퓨터 대부분이 IBM PC 호환기종으로 보급된 것도 이런 이유에서 비롯되었다고 할 수 있다.

(3) CPU

① 두 유명 가문의 숙명적 대결

컴퓨터회사의 선전광고를 보면 반드시 00 비트의 XXXX CPU라는 말부터 시작한다. CPU는 컴퓨터의 가장 중요한 부품으로 그 컴퓨터가 할 수 있는 일의 범위가 대부분 이것에 의해 결정된다. 보통 16비트 CPU라든가 32비트 CPU라고 하는데 비트수가 크면 클수록 계산도 빠르고 성능도 좋다. 그러나 여러 회사에서 다양한 컴퓨터를 만들고 있지만 컴퓨터 내부를 열어 보면 CPU는 대부분 인텔사의 80계열이나 모토롤라사의 68계열 두가지중 하나를 사용하고 있다. 즉 퍼스널컴퓨터가 여러가지 상표로 판매되고 있지만 알고 보면 가장 중요한 CPU는 대부분이 두 유명한 회사 중의 하나라는 말이 된다.

마이크로프로세서 다시 말해 퍼스널

컴퓨터에 사용되는 CPU는 워낙 복잡하고 또 그동안 수많은 기술이 축적되었기 때문에 이제는 커다란 컴퓨터 회사조차 새로이 CPU를 설계하기 보다는 오래전부터 CPU를 대량으로 만들고 있는 두 회사중 하나에게 부탁하거나 혹은 직접 CPU를 생산하더라도 기술료를 지불하고 모방 생산하는 실정이다.

CPU가 두개의 커다란 가문으로 구별되는 내력을 알고 나면 컴퓨터 광고만 보아도 그 컴퓨터의 특징이나 장단점은 대강 짐작할 수 있게 된다.

● 4비트 마이크로프로세서의 등장

퍼스널컴퓨터가 나오기 오래 전부터 이미 대형컴퓨터들이 사용되고 있었다. 그러나 대형컴퓨터를 만들던 회사들은 작은 마이크로컴퓨터를 개발하는데 그렇게 큰 관심이 없었다. 오히려 휴대용 전자계산기를 제작하는 회사들이 성능개선과 소형화의 치열한 경쟁을 하는 중에 마이크르프로세서가 만들어지고 이것이 퍼스널컴퓨터로 발전했다고 할 수 있다.

요즘은 국민학생도 산수 숙제를 하는데 휴대용 계산기를 사용할 정도로 크기가 작아지고 흔해졌지만 1960년대 말에만 해도 휴대용 계산기의 크기를 줄이기 위한 경쟁이 치열했다. 특히 소형제품을 만드는 것이 장기인 일본의 회사들이 계속적으로 보다 작은 신제품을 발표했다.

그 중에 하나인 비지콘사는 계산, 제어, 기억기능의 전자회로를 모아서 하나의 IC안에 집어 넣은 LSI를 만들어 여러가지 모델의 휴대용 계산기에 공통으로 사용하려는 아이디어를 갖게 되었다. 당시 일본의 기술수준으로는 생산이 어려워서 IC 제작 기술이 앞서 있는 미국의 여러 IC회사를 찾아 다니며 어렵게 개발을 부탁한 끝에 세계 최초의 마이크로프로세서를 만들어 냈다. 그러나 이것은 아직 완전한 컴퓨터를 만들 수 있는 것은 아니었고 휴대용 계산기를 위한 것이었다.

● 8비트 CPU의 두 가문의 등장

1972년 미국의 인텔사는 4004를 개량해서 한번에 8비트씩 계산할 수 있는 보다 고성능의 휴대용 계산기를 만들기 위한 8008을 발표했다. 이 8008은 컴퓨터를 만들 수 있을 정도의 능력이 있었고, 곧 이어서 1973년 다시 이를 개량한 본격적인 마이크로프로세서 8080 CPU시대가 열리게 되었다. 한편 1974년 IC의 대 메이커 모토롤라사는 대형컴퓨터의 회로를 소형화해서 6800 CPU를 발표했다.

인텔사의 80계열 CPU가 휴대용 계산기로 출발해서 점차 성능이 확대 되었다면, 모토롤라사의 68계열은 본격적인 대형컴퓨터를 소형으로 축소한 CPU였다. 말하자면 80계열은 어려운 집안에서 태어나 노력한 끝에 자수성가하여 한 가문을 이루었다면, 68계열은 넉넉한 집안 출신으로 체계적인 교육을 받고 성장한 셈이다. 서로가 상대적인 장단점이 있어 컴퓨터를 설계하는 기술자들도 각자의 취향에 따라 어느 편이 우수한가에 대한 논란이 계속되고 있다.

간단하게 특징을 말한다면, 80계열은 CPU 내부기능 특히 레지스터가 많고 CPU 내부작업이 신속하다. 반면에 68계열은 대형컴퓨터 출신답게 잘 정리된 명령계통을 갖고 있으며, CPU 내부의 레지스터는 적으나 CPU밖에 있는 메모리와 연결이 좋다.

● 계속되는 편싸움

컴퓨터를 만드는 회사들도 컴퓨터에 들어가는 CPU를 한번 정하면 바꾸기가 어렵다. 왜냐하면 몇배나 강력한 CPU가 새로 개발되었다고 해도 전에 사용하던 같은 계열의 CPU의 명령어를 대부분 그대로 사용할 수 있어 그전의 컴퓨터에서 사용하던 프로그램을 새로운 컴퓨터에 쉽게 고쳐 사용할 수 있기 때문이다.

따라서 CPU의 경쟁은 인텔사와 모토롤라사 만의 문제가 아니라 수많은 컴퓨터회사까지도 양분되는 편싸움의 형태가 되고 있다.

8비트에서 32비트까지 계속되는 두 CPU 가문의 경쟁은 80계열에서는 8080, Z80, 8088, 8086, 80286, 80386으로 계속 자손들이 번창하였고, 68계열에서는 6800, 6502, 6809, 68000, 68020, 68030의 막강한 시리즈로 대를 이어 계속되고 있다.

한편 어쩔 수 없이 이 싸움에 끼어들게 되는 컴퓨터회사들도 자연히 편이 갈라지게 되었다. 최초의 승부는 8비트 컴퓨터부터 시작되었는데 대형컴퓨터의 원조인 미국에서 6800을 개조한 6502 CPU의 APPLE Ⅱ가 압도적인 우세를 차지했고, 휴대용 계산기의 나라 일본에서는 계산기용 CPU 8008에서 발전한 Z80을 사용한 PC-8001이 시장을 휩쓸었다.

2차 승부전이 된 16비트에서는 컴퓨터 업계의 거인 미국의 IBM사가 시장에 새로 참여하자 판도가 크게 변하게 되었다. IBM이 CPU로 택한 8088은 16비트 CPU이지만 8비트의 데이터버스에 16비트의 레지스터를 갖고 있어 8비트의 특성이 많이 남아 있는 것으로 8비트 컴퓨터에서 축적된 자원을 쉽게 사용할 수 있었다.

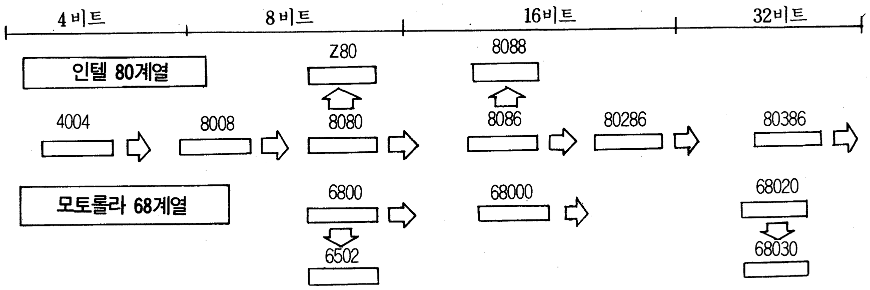
반대로 8비트 컴퓨터시장을 장악해 오던 애플사가 IBM PC에 대항하기 위해 개발한 16비트 컴퓨터 리자와 매킨토시에 68000을 사용하였는데 이것은 16비트이지만 내부는 32비트 레지스터가 있다. 기술적으로는 매킨토시 쪽이 더 고급이라고 할 수 있지만 그때까지 사용해 오던 8비트 컴퓨터의 자원을 그대로 사용하기 어려워 대부분 새로 개발해야만 했다. 더우기 애플사는 그 회사가 만드는 컴퓨터 APPLE Ⅱ가 최고의 인기를 누렸지만 수많은 회사가 모방품을 만들어 싸게 파는데 화가 치밀어 모방할 수 없도록 극도로 폐쇄적인 정책을 폈다. 반대로 IBM은 누구나 마음만 먹으면 쉽게 모방할 수 있을 정도로 자세한 기술 자료를 공개해 값싼 모방품이 쏟아져 나왔다.

IBM은 마침내 퍼스널컴퓨터 시장에서 애플사를 누르고 16비트 컴퓨터의 새로운 표준이 되었다. 역시 온세상 돈을 혼자 모두 벌려는 욕심보다는 적당히 조그만 회사들도 살아갈 수 있도록 선심을 쓴 쪽이 결과적으로 이기게 된 셈이다.

그래도 애플사의 컴퓨터 매킨토시는 비록 IBM PC에게 챔피언 자리를 빼았겼지만 고급 성능을 찾는 소비자들에게 인기가 꾸준히 높아져서 이제는 무시 못할 수준까지 올라와 IBM PC와 대결을 계속하고 있다. 그리고 애플사를 설립한 사장이면서도 무모할 정도의 기술투자를 고집하다 회사에서 밀려난 스티브 잡스가 야심적으로 발표한 NeXT는 68계열인 32비트 68030을 CPU로 사용하여 환상적일 정도의 놀라운 성능을 보여주고 있어 화제의 대상이 되고 있다.

16비트에서의 대결은 80계열의 우세로 판정이 났지만 본격적인 대형 컴퓨터의 수준인 32비트 컴퓨터의 주도권을 놓고 68계열 CPU를 사용하는 컴퓨터들이 일대 반격을 하고 있는 상황이다(그림 2 참조).

〔그림 2〕 CPU 계통도



8비트 CPU Z80은 인텔에서 독립한 기술자들이 세운 자일로그사의 제품이다. 원형인 인텔의 8080과 동일한 명령을 사용하고 기능을 보다 강화하여 많은 컴퓨터에 이용되고 있다.

6502는 모스테크놀리지사의 제품이나 68계열의 구조와 거의 동일하고 연산속도를 강화하였다. 애플 컴퓨터에 사용되어 8비트 컴퓨터중 전세계에서 가장 인기있는 것이 되었다.

(4) 메모리

CPU가 직접 사용하는 메인 메모리는 필수적인 사항이다. 불과 수년전 만해도 퍼스널컴퓨터는 64 킬로바이트의 램을 기준으로 하였다. 8비트 컴퓨터는 메모리의 번지수를 직접 지정할 수 있는 최대의 크기가 64 킬로바이트로 이 정도면 개인이 사용하기에 충분하다고 생각했다. 그러나, 컴퓨터를 사용하는 사람들이 보다 편하게 사용하기 위해서는 많은 메모리가 필요하다.

컴퓨터 화면은 가로 세로를 점으로 연결하여 문자나 그림을 그린다. 화면에 구성하는 점의 갯수를 많이 하면 할수록 그림이나 글자가 선명해지고 보기에도 깨끗하지만 여기에 필요한 메모리는 기하 급수적으로 늘어난다. 요즘은 화면을 고급스럽게 처리하기 위해서 비디오 전용으로 만도 256 킬로바이트가 사용된다.

16비트 컴퓨터에서는 메모리가 보통 기본적으로 512킬로바이트이상이 되고 있지만 한단계 올라간 32비트 컴퓨터로 가면 최소 1메가에서 16메가바이트를 사용한다. 하여간 컴퓨터는 CPU의 비트수가 많고 메모리가 크면 클수록 사용하는 사람이 신경 쓸 일이 적어지고 편리하게 사용할 수 있다. PC

〈용어해설〉

●**데이터베이스(Data Base)**
하나이상의 데이터모임으로 그 내용을 고도로 구조화하여 검색 또는 갱신할 수 있게 하는 것으로 몇개의 데이터 화일을 조직적으로 통합하여 데이터 항목의 중복을 없애고 데이터를 구조화하여 기억시킨 것이다.

●**전문가시스템(Expert System)**
전문가가 갖는 지식을 지식의 데이터베이스로 저장하여 비전문가라도 그 지식을 이용할 수 있도록 한 시스템

●**CPU(Central Processing Unit)**
컴퓨터 구성요소의 명령과 해독과 실행을 제어하는 회로를 가지며, 통상 연산장치와 제어장치로 구성된다.

●**마이크로프로세서(Micro Processor)**
마이크로 컴퓨터의 CPU를 말하기도 하며 마이크로 컴퓨터를 가리키는 경우도 있다. 마이크로 프로세서는 연산부, 제어부, 레지스터부의 3개의 부분으로 구성된다. 연산부는 산술연산 및 논리연산을 하는 것으로 산술논리 유니트, 어큘레이터, 어큐물레이터래치, 임시 레지스터, 플래그 플립플롭 등으로 되어 있다. 제어부는 CPU의 동작을 제어하는 것으로 명령 레지스터, 명령 디코더, 머신 사이클 엔코더, 타이밍과 제어부 등으로 되어 있다. 레지스터부는 CPU의 내부 메모리 기능을 가지며 B, C, D레지스터 등의 범용레지스터나 W, Z레지스터 등의 일시 레지스터, 그리고 인크리먼트/디크리먼트와 어드레스 래치나 레지스터 선택 및 멀티플렉서 등으로 되어 있다. 또 어큐물레이터도 A레지스터로 불리워 범용레지스터에 속하나 이것은 연산부에 속한다.

●**IC(Integrated Circuit)**
반도체만으로 전자회로를 구성하려고 하는 시도와 고도의 반도체 처리기술에 의해 생긴 회로로 트랜지스터, 다이오드, 저항, 경우에 따라서는 콘덴서 등의 회로소자를 서로 분리할 수 없도록 집정상태로 결합하여 하나의 반도체 기판에 매입한 초소형 구조의 회로.

어떻게 시작하나요 5
입·출력 I/O의 재지정

이달에는 입·출력 I/O의 재지정에 대해서 알아 보자. 원래 이 기능을 비롯한 파이프(Pipe)명령, 그리고 필터(Filter)기능들은 UNIX 운영체제에서 이용되고 있었던 것인데 이제는 MS-DOS에서도 쓰일 수 있게 된 것이다.

여기서는 대부분 배치화일을 이용해서 일련의 기능을 실행하거나 저장하게 되는데 지난달에 소개했던 배치화일 작성에 관한 내용을 참고하면 이해하는데 별 어려움이 없을 것이다. 디렉토리에 어떤 배치화일이 있을 때 그것이 어떤 기능을 수행하고 있을까 하고 의문이 생기면 주저말고 'TYPE'문을 이용하면 된다. 새로운 프로그램이 들어 있는 디스켓일지라도 그속의 배치화일만 제대로 분석하면 실행에 관한 전반적인 정보를 얻게되는 경우가 대부분이다.

컴퓨터에 데이터를 입력시키고 출력시키는 장치(Input Output Device : I/O 디바이스라 한다.)에는 여러 종류가 있다. 그러나 컴퓨터를 사용하는 사람들이 이런 장치들을 모두 구비하고 있기를 바란다는 것은 무척 힘든 일이다. 그래서 컴퓨터를 이용할 때 기본적으로 꼭 필요한 입력장치와 출력장치만을 미리 표준형으로 정해서 쓰고 있는데 이것을 표준 입·출력장치라고 부른다. 즉 표준 입력장치는 키보드로 되어 있고 표준 출력장치는 모니터(화면)로 되어 있다.

이 말은 다시 표현하면 컴퓨터를 사용할 때 키보드와 모니터는 반드시 장만해야 하는 주변장치라는 뜻이 되기도 한다. 설마 독자들 중에서 컴퓨터(여기서는 본체를 뜻함)를 구입하면서 다른 주변장치는 몰라도 키보드와 모니터없이 쓰고자 하는 사람은 없으려라 본다.

보통은 본체, 키보드, 모니터를 기본 시스템으로 사용하다가 형편이 되는대로 디스크 드라이브를 비롯하여 프린터 등의 주변장치를 구입하고 있는데 바로 이럴 때 입·출력 I/O재지정을 통해 입력 데이터를 키보드로 받다가 디스크 드라이브에서 받게 하거나, 출력 데이터를 모니터 화면으로만 출력하다가 대신 프린터로 출력하게 하는 것이다. 이때 DOS는 단지 논리 디바이스(표준 입·출력)의 매핑(Mapping) 만을 다루기 때문에 실제 어떤 물리적인 장치가 접속되어 있는지에 대해서는 상관을 않게 된다.

표준 에러메시지 또한 화면으로 매핑되는 논리 디바이스라고 볼 수 있다. 이 논리 다바이스는 시스템 프로그램이나 DOS의 실행에서 에러 상황을 알리기 위해 사용되는데 이 표준 에러메시지는 사용자들이 볼 수 있는 형태로 출력되어야 하므로 재지정을 할 수 없게 만들어졌다. (에러메시지 출력을 구태여 화면을 제외하고 다른 주변장치로 출력할 필요는 없기 때문이다.) 다음 경우를 살펴 보자.

```
DIR A:
```

이 명령을 실행하면 A디스크의 현재 디렉토리가 화면에 나타날 것이다. 이번에는 다음과 같이 입력해 보자.

```
DIR A: > NUL
```

이번에는 화면에 아무것도 출력되지 않는다. 즉 DOS에 의해 디렉토리를 출력할 디바이스가 모니터화면에서 'NUL디바이스'로 바뀌어졌기 때문이다. 이번에는 A드라이브의 도어를 열고 위의 명령을 다시 한번 실행시켜 보기 바란다. 물론 에러가 발생할 것이다. 그런데 지금 DOS는 출력 디바이스를 NUL로 알고 있는 상태이긴 하지만 표준 에러메시지는 재지정이 안되기 때문에 다음과 같이 화면을 통해 출력됨을 볼 수 있다.

```
Not ready error reading drive A
Abort, Retry, Ignore?
```

앞에서 '>'을 이용하는 것은 'NUL디바이스'로 할당하는 의미인데 이 형태의 명령을 쓰는 주된 목적은 통신포트들 중에서 어느 한 부분을 입력 또는 출력장치로 재지정을 하기 위한 것으로 볼 수 있다. 이들 포트로는 COM1, COM2, LPT1, PRN, NUL 등이 있으며 이들은 다른 컴퓨터와 통신을 하거나 터미널로서 자기의 컴퓨터를 제어할 수 있게 해 준다. 다유 경우를 통해 쉽게 이해해 보자.

□주의
DOS의 'DIR'명령을 출력 디바이스의 재지정과 함께 사용하여 독자적인 디렉토리 화일을 만들 수 있다.

DIR명령에 의해 디렉토리 정보가 디스플레이된 화면은 표준 출력장치로서 일시적인 것이다. 그렇지만 출력을

적당한 화일이름으로 재지정하면 디렉토리 정보를 독립적인 화일로 만들 수 있고 필요로 할 때 유용하게 쓰일 수 있다.

```
A>DIR > DIRFILE.EXT
A>TYPE DIRFILE.EXT
A>PRINT DIRFILE.EXT
```

마찬가지로 디렉토리를 프린터로 직접 출력하고자 할 때는 다음과 같이 재지정을 이용한다.

```
A>DIR > LPT1 [PRN 또는 LP2]
```

다른 경우로써 키보드를 통해 입력되는 내용으로 구성되는 화일을 만들 수 있는데 이때 키보드를 통해 응답하는 방법은 두가지가 있다.

① 일련의 문자를 기억할 수 있으며 〈ENTER〉 키가 입력될 때까지의 데이터가 기억된다.
② 단지 한 문자를 받아 들임으로써 그에 따르는 어떤 효과를 얻는 방법이다. 한 예로 DOS상의 'strike any key when ready' 메시지를 보면 같은 줄에서 아무키든지 누르기만 하면 정해진 동작을 실행해 버리는데 이것을 'INKEY'타입의 데이터라고 부른다.

이 사실을 조금만 응용하면 매일 반복되는 컴퓨터의 조작을 좀 더 세련되게 할 수 있다. 두 종류의 응답을 이해하기 위해 COMP명령을 예로 들어 보자. 이 COMP명령은 2개의 화일을 비교하기 위한 명령으로 다음과 같이 3단계의 응답을 필요로 한다.

① 화일이름 1 〈ENTER〉
② 화일이름 2 〈ENTER〉
③ 다른 화일도 비교하는지의 여부를 묻는 'Y' 또는 'N'

만일 2개의 플로피디스크 드라이브에 있는 AUTOEXEC.BAT화일을 비교하고 싶은 경우는 다음과 같이 키보드를 통해 응답화일 COMPIN을 만든다.

```
COPY CON: COMPIN
A:AUTOEXEC.BAT ↵
B:AUTOEXEC.BAT ↵
N —— 〈F6〉키를 누르기 전에 'N'을 입력한다
^Z
  1 file(s) copied
```

이제는 다음과 같이 키보드의 입력대신 방금 작성한 응답화일로 재지정한 상태로 COMP명령을 실행시킬 수 있다.

```
COMP < COMPIN
```

이때 'N'(No more than)이나 〈ENTER〉 의 입력을 다음과 같이 별개로 작성하여 응용할 수도 있다.

```
COPY CON: NO
N ↵
^Z ——— 함수키 〈F6〉을 누른다

COPY CON: CR
↵ ———〈ENTER〉키를 누른다
^Z ——— 함수키 〈F6〉을 누른다
```

□주의
사용자가 입력화일을 미리 작성하여 프로그램을 자동적으로 실행시킬 때 입력화일 속에 모든 응답을 제공하지 않으면 실행이 중지되어 버릴 수 있다.

위의 경우로써 COMP명령을 실행하기 위해서는 '화일이름 1', '화일이름 2' 그리고 'N'과 'Y'의 응답을 필요로 한다. 그럼 응답화일에서 임의의 응답을 삭제시키고 어떤 일이 일어나는지에 대해 살펴 보기로 하자.

```
COPY CON: COMPIN
A:AUTOEXEC.BAT
B:AUTOEXEC.BAT
^Z
  1 file(s) copied
```

다음은 키보드의 입력을 위의 응답화일로 재지정한 후 COMP명령을 실행시킨다.

```
COMP < COMPIN
```

예상대로 DOS는 실행을 중지시키고 커서가 깜박거리는 상태가 되어 버린다. (이때는 할 수 없이 〈Ctrl〉-〈Break〉키를 누르거나 〈Alt〉-〈Ctrl〉-〈Del〉키를 눌러 웜부트(Warmboot)를 시켜야 한다.)

□주의
출력디바이스를 '〉'로 재지정했을 때 자칫하면 기존에 있었던 화일을 잃게 되는 경우가 있으므로 주의한다.

입 · 출력 재지정을 할 때 키를 잘못 입력하면 큰 실수를 하는 경우가 있다. 예를 들어 이미 존재하는 화일이름으로 출력을 재지정해 버리는 경우를 보자. 당연히 새로운 출력으로 기존의 내용을 잃게 될 것이다. 또 다른 경우로는 출력을 다른 프로그램으로 파이프하고자 할 때 파이프 문자인 '|'를 '〉'로 잘못 입력하여 데이터를 옮기는 대신에 프로그램 자체가 오버라이트되어 버리는 경우이다. (보통 FIND.EXE, MORE.COM 그리고 SORT.EXE프로그램 등에서 이런 실수를 범하기 쉽다.)

디렉토리의 정보가 방대할 경우에 디렉토리의 내용을 화면단위로 차례차례 볼 수 있도록 해주는 명령이 MORE.COM이다. 이 명령을 이용할 때는

```
DIR ¦ MORE.COM
```

과 같이 해야 하는데 실수로

```
DIR > MORE.COM
```

으로 입력했다면 화면단위의 디렉토리를 볼 수 없을 뿐만 아니라 이 디렉토리의 내용이 MORE.COM화일에 쓰여지게 된다. 이때는 DOS백업 디스켓으로부터 MORE.COM을 새로 복사해 넣어야 할 것이다. (실제는 MORE.COM이라고 입력하는 대신에 MORE라고만 지정하는 경우가 대부분이다. 이때는 MORE라는 별개의 화일만 만들어질 뿐이다.)

축적되는 화일이 지워지는 것을 막을 수도 있다. 주기적으로 계속 처리되고 축적되어 가는 데이터 화일(예 : 증

권분석 프로그램, 물량관리 프로그램 등)이 있다고 할때, 하루에 처리된 데이터를 구분하기 위한 'Logoff'배치화일을 다음과 같이 만들 수 있다.

```
COPY CON: LOGOFF
TYPE DAILY.TRN >> WEEKLY.TRN
^Z
```

그래서 주말마다 이 WEEKLY.TRN 화일을 지우고 월요일에 다시 WEEKLY. TRN을 만들기 위해 LOGOFF를 실행시킨다. 그런 후 화요일부터 금요일까지 매일 처리되는 데이터를 계속 WEEKLY.TRN화일의 끝에 덧붙이기 위해 LOGOFF를 실행하는 것이다.

□주의

배치화일 레벨에서는 입·출력 재지정을 할 수 없다.

DOS의 명령이나 배치화일내의 프로그램에서는 입·출력 재지정이 가능하지만 배치화일을 실행시키는 동일 라인에서는 같은 재지정을 할 수 없다. 그 이유는 배치화일이 재지정 연산자인 '>'와 화일이름을 파라미터로 간주하지 않기 때문인데 다음 경우는 출력재지정과 함께 사용된 배치화일의 예를 보여 주고 있다.

```
COPY CON: REDIRECT.BAT
COPY AUTOEXEC.BAT TEMP > NUL
^Z
```

위의 배치화일을 실행시켜 보면 '1 File(s) copied'라는 메시지가 화면에 나타나지 않는다.

```
REDIRECT
COPY AUTOEXEC.BAT TEMP > NUL
```

이번엔 내부에 재지정을 하지 않는 배치화일을 만들어 보자.

```
COPY CON: REDIRECT.BAT
COPY AUTOEXEC.BAT TEMP
^Z
```

이 배치화일을 다음과 같이 출력화일을 'TESTOUT'로 재지정한 후 실행시킨다.

```
REDIRECT > TESTOUT
```

이번엔 '1 File(s) copied'라는 메시지가 화면에 나타났을 것이다. 즉 DOS는 배치화일 이름뒤에 지정한 재지정을 무시해 버린다는 것을 알았을 것이다. (배치화일도 재지정과 함께 쓸 수 없는 프로그램중의 하나이다.)

□주의

원하지 않는 출력을 없애는 경우는 'NUL 디바이스'로 출력 재지정을 하면 된다.

NUL은 불필요한 데이터를 처리하기 위해 마련한 디바이스이다. 배치화일의 'ECHO OFF'명령을 가지고서 제대로 삭제되지 않는 DOS명령의 출력 등을 나타나지 않게 하는데도 바로 이 NUL디바이스를 이용하면 된다.

예를 들어 매일 사용해야 하는 표준 배치화일이 있을 때 DOS명령어를 비롯해서 관련 주석문 등 메시지를 포함하고 있으면 이런 메시지는 NUL로 재지정하지 않는 한 실행할 때마다 매번 나타나는 것을 볼 수 있다(**리스트** 1 참조).

만일 DOS명령의 출력을 NUL로 재지정했다면 DOS프롬프트는 화면상에 나타나지 않은 상태에서 사용자의 응답을 기다리게 된다.

예를 들어 FORMAT 명령의 출력을 NUL로 재지정했을 경우를 보자(디폴트 드라이브는 C).

```
FORMAT > NUL
```

위의 명령을 실행하면 커서가 빈 라인에서 깜박거리고 있을 것이다. 사실 FORMAT명령이 이 순간에도 다음과 같은 메시지를 계속 내보내고 있다.

```
WARNING, ALL DATA ON NON-REMOVABLE
         DISK
DRIVE C: WILL BE LOST!
Proceed with Format (Y/N)?
```

그러나 이 메시지는 계속 NUL디바이스로 보내지고 있기 때문에 사용자로서는 볼 수 없게 된 것 뿐이고 실제로 'Y' 또는 'y'를 눌렀다면 명령은 실행되고 따라서 하드디스크의 모든 화일을 지우게 된다. 그러므로 화면 출력을 NUL로 재지정할 때는 매우 조심해서 실행시켜야 한다는 것을 명심하자.

□주의

현재의 PATH를 기억한 후 다른 작업을 하다가 다시 원래의 PATH로 자동 복귀시킬 수 있다.

어떤 작업을 하다가 다른 프로그램을 이용하기 위해서는 그 프로그램의 실행환경에 알맞게 패스를 설정해야 한다. 그런데 프로그램 실행이 끝나면 다시 원래의 패스로 돌아와야 하는데 이것은 간혹 귀찮은 일이 되곤 한다. 바로 이 경우에 기존의 패스를 홀딩화일(Holding File)로 기억시킨 후 나중에 다시 불러들일 수 있다. 현재의 패스를 기억시키기 위해서 다음과 같이 입력해 보자.

```
PATH > SAVEPATH.BAT
```

만일 현재의 패스가 'A:\DIR1;A:\DIR2'이라면 다음과 같은 새로운 화일 SAVEPATH.BAT가 생성된다.

```
PATH=A:\DIR1;A:\DIR2
```

마찬가지 방법으로 임의의 디렉토리에서 현재의 패스를 저장한 후 다른 프로그램을 실행하다가 다시 원래의 패스로 되돌아가기 위해 **리스트** 2와 같은 배치화일을 생각할 수도 있다.

다른 디렉토리에 있는 프로그램을 이용하기 위해서는 해당 디렉토리로 제어를 옮긴 다음 원하는 프로그램을 실행시키는 것은 당연한 일이라 하겠다. 그러나 문제는 프로그램 실행이 끝나면 다시 원래의 디렉토리로 돌아오는 작업을 매번해야 한다는 사실이다. 이것 또한 앞에서 설명한 홀딩화일을 이용하여 자동적으로 원래의 디렉토리

〔리스트 1〕 배치화일에서의 메시지

```
REM **************************************
REM *          NULEXAMP.BAT              *
REM **************************************
REM
ECHO ON: Display DOS command output.
REM
     COPY AUTOEXEC.BAT
REM **************************************
REM *  With ECHO ON, both the COPY       *
REM *  command and the message display   *
REM *                                    *
REM *  COPT AUTOEXEC.BAT                 *
REM *  File cannot be copied onto itself*
REM *  0 file(s) copied                  *
REM **************************************
REM
ECHO OFF: Do not display DOScommand.
ECHO *
ECHO * "COPY AUTOEXEC.BAT" is next.
ECHO *
        COPY AUTOEXEC.BAT
```

```
ECHO ON: Resume display of commands.
REM *
REM *************************************
REM * Though you suppress display of    *
REM * the COPY command with ECHO OFF,   *
REM * the COPY message still display    *
REM *                                   *
REM *File cannot bo copied onto itself *
REM *   0 File(s) copied                *
REM *************************************
REM *
REM *   "COPY AUTOEXEC.BAT > NUL"  next.
REM *
ECHO OFF: Do not display DOS commands.
        COPY AUTOEXEC.BAT > NUL
REM *
REM *************************************
REM * By redirecting to NUL, YOU        *
REM * suppress the COPY message also.   *
REM *************************************
```

〔리스트 2〕 EXECPATH.BAT

```
COPY CON EXECPATH.BAT ←——— 배치화일 생성
PATH > SAVEPATH.BAT ←——— 현재의 패스화일 생성
PATH=C:\DIR5;C:\DIR6 ←— 프로그램 실행을 위한 새로운 패스 지정
PROGRAM ←——— 프로그램 실행
SAVEPATH ←——— 원래의 패스로 돌아옴
^Z ←——— 함수키 〈F6〉을 누른다
```

〔리스트 3〕 EXECHDIR.BAT

```
COPY CON EXECHDIR.BAT ←——— 배치화일 생성
TYPE CHDIR > SVCURDIR.BAT ←——— 새 CHDIR명령 생성
CD >> SVCURDIR.BAT ←——— CHDIR에 현재의 디렉토리를 첨가
CD A:\DIR3 ←——— 프로그램 실행을 위한 새로운 디렉토리 세트
PROGRAM ←——— 프로그램 실행
SVCURDIR ←——— 원래의 디렉토리로 돌아옴
^Z ←——— 함수키 〈F6〉을 누른다
```

로 옮기는 방법에 대해 알아 보자. 현재의 디렉토리를 홀딩화일에 넣기 위해서 다음 명령을 입력한다.

```
CD > SVCURDIR.BAT
```

이와 마찬가지로 현재의 디렉토리가 'A:\DIR1'이라면 다음의 디렉토리 이름을 갖는 새로운 화일 'SVCURDIR.BAT'가 생성된다.

```
A:\DIR1
```

그렇지만 현재의 디렉토리에서 'A:\DIR1'로 바꾸기 위해서는 'CHDIR'명령이 필요한데 그 완전한 형식은 다음과 같다.

```
CHDIR A:\DIR1
```

즉 이와같이 완전한 명령을 얻기 위해서는 단지 'CHDIR'을 가지고 이미 존재하는 배치화일에 현재의 디렉토리명을 덧붙인다. 명령의 일부를 나타내면 다음과 같다.

```
COPY CON CHDIR
CHDIR ↵ ←{ 공백을 띄고 〈F6〉키를 누른 다음
            〈ENTER〉키를 누른다
^Z
```

다음, 임의의 디렉토리에서 현재의 디렉토리를 기억하고 새로운 디렉토리를 만들고 원하는 프로그램을 실행시킨 다음 원래의 디렉토리로 돌아오기 위한 배치화일을 만든다(리스트 3).

지금까지 입·출력의 재지정에 관한 사항을 간략히 알아 보았다. 다음달에는 글 첫머리에서 언급한 파이프와, 필터기능에 대해 설명하고자 한다. PC

화일의 결합과 분리

개인이 하기에는 많은 양의 워드프로세싱 작업을 해야 할 때 여러 사람이 이 작업을 나누어 할 수 있다면 매우 신속하게 일을 마칠 수 있게 된다. 이렇게 작업을 분담하게 된다면 자연히 다수의 화일이 생기게 된다. 그러나 각각 분리되었던 작업은 다시 합하여야만 완전한 일을 이룰 수 있는 것처럼 이 각각의 화일들을 통합하여야만 한다. 그러면 화일을 합치려면 어떻게 해야 할까? 그리고 이 합쳤던 화일을 다시 분리해야 할 때는 어떻게 해야 할까? 이럴때 간단한 명령어를 이용하여 일을 효과적으로 처리할 수 있는 방법을 알아보자.

화일의 결합

화일을 결합하는 방법은 여러 상태에서 할 수 있으나 예를 들어 도스 버전3.3과 터보파스칼 4.0상태에서 어떻게 하는지 알아보자.

(1) 도스3.3에서의 화일 결합

도스3.3에서는 COPY 명령어를 사용한다. 즉, COPY 명령어에 결합하려는 화일을 +로 열결하면 된다. 예를 들어 A 드라이브에 P.TXT와 C.TXT화일이 있고 P.TXT에는 My name is Dooly라는 내용이 C.TXT에는 I love Personal Computer 이라는 내용이 들어있다고 하자. 이 두화일을 결합하고 싶다면 다음과 같이 하면 된다.

```
C>COPY A:P.TXT+A:C.TXT A:PC.TXT
```

이렇게 하면 A 드라이브의 P.TXT 화일과 C.TXT 화일이 결합하여 A 드라이브의 PC.TXT라는 화일로 저장된다. 화면 1에 그 과정을 나타내었다.

만약 새로운 화일명을 주지 않으면 결합하려는 첫번째 화일로 결합된 화일이 저장되고 이전의 내용은 지워지게 된다.

이러한 화일의 결합은 단일 화일 뿐만 아니라 와일드카드 문자를 사용하여 여러개의 화일을 결합할 수도 있다. 예를 들어 다음과 같이

```
COPY *.LST+*.REF COMB.PRN
```

하면 LST라는 확장명을 가지는 모든 화일과 REF라는 확장명을 가지는 모든 화일이 COMB.PRN이라는 화일로 결합되어 저장된다.

(2) 터보파스칼상태에서의 화일결합

터보 파스칼 4.0에서는 에디터에서 외부화일을 BLOCK BUFFER로 읽어 들여 현재 작업중인 화일의 중간에 삽입하는 기능이 있다. 이 기능을 이용하여 화일을 결합시킬 수 있다.

우선 터보 파스칼 메인 메뉴에서 새로운 화일을 생성시킨 후 에디터 모드로 들어간다. 화면 최상단에 커서를 옮긴 후 Ctrl-K R(Ctrl과 K를 동시에 누른 후 R을 누름)을 누르면

```
Read Block From File
```

와 같은 메시지가 나온다. 이때 결합하려는 첫 화일의 이름을 입력하면 화일의 내용이 로드된다.

다음에는 첫화일의 맨 끝줄 아래로 커서를 이동시킨 후 다시 Ctrl-K R로 블록 로드기능을 사용하면 아래 라인에 내

〔화면 1〕 도스에서의 화일 결합

```
C:\DOS\DOS3_3>COPY A:P.TXT+A:C.TXT A:PC.TXT
A:P.TXT
A:C.TXT
        1 File(s) copied

C:\DOS\DOS3_3>A:

A:\>DIR/W

 Volume in drive A has no label
 Directory of  A:\

GCG          CAL    BAK  CAL          MERGE       PC111
PC01         PC03        P      TXT   PC04   BAK  PC04
C      TXT   MERGE  BAK  PC     TXT
       13 File(s)    222208 bytes free

A:\>TYPE PC.TXT
My name is Dooly.
I love Personal Computer.

A:\>
```

용이 첨가된다. 다시 이러한 작업을 화일을 모두 결합할 때까지 반복하고 모두 마치게 되면 메인메뉴로 빠져 나가 저장한다.

터보 파스칼에서 화일을 결합하게 되면 편집할 수 있는 방법도 제공한다는 장점이 있지만 도스상태에서 하는 것보다 작업이 많다.

화일의 분리

이제 결합되었던 화일을 분리하여 보자. 화일의 분리는 COPY 명령어로는 할 수 없고 터보 파스칼의 BLOCK WRITE 기능으로는 가능하다.

예를 들어 PC.TXT를 P1.TXT과 C1.TXT로 분리하고자 할 때 다음과 같이 한다.

터보 파스칼의 메인메뉴에서 PC.TXT를 로드하여 에디터 모드로 들어간다. 에디터 모드에서 분리하고자 하는 내용의 맨앞에서 Ctrl-K B(Begin of block) 를 입력하고 분리하려는 부분의 맨 뒤에서 Ctrl-KK(End of BLOCK)를 입력하면 내용에 해당하는 부분이 역상문자로 표시된다.

이때 Ctrl-K W를 입력하면 다음과 같은 내용이 출력된다.

```
Write Block To File
```

여기에 A : P1.TXT를 입력하면 A 드라이브에 BLOCK의 내용을 가진 P1.TXT 화일이 생성된다. 화면 2에 이 과정을 나타낸다.

〔화면 2〕 터보 파스칼 4.0에서의 화일 분리

```
     File      Edit      Run      Compile      Options
═══════════════════════════════════════ Edit ═══════════
^Kw   Line 8     Col 18  Insert Indent      A:PC.TXT
   ┌── Write Block To File ──┐
   │a:p1.txt                 │
   └─────────────────────────┘

My name is Dooly.
I love Personal Computer.

──────────────────────────────────── Output ─────────────

C:\PASCAL\T_PASCAL>turbo


C:\PASCAL\T_PASCAL>turbo

F1-Help  Esc-Abort
```

이제 Ctrl-K Y를 입력하여 블록을 삭제시킨다. 이것은 블록이 남아 있으면 혼돈하기 쉽기 때문이다. 이렇게 하여 모든 내용이 분리되면 메인메뉴에서 PC.TXT를 저장하지 않고 빠져 나온다. 만약 저장하게 되면 이전의 PC.TXT가 백업되어 버린다.

지금까지 화일의 결합과 분리에 대하여 간단하게나마 알아 보았다. 터보 파스칼의 에디터는 이런 기능이외에도 BLOCK Print 기능이 있어 Ctrl-K P를 입력하게 되면 블럭의 내용이 프린트된다. 그리고 터보 파스칼 3.0에서는 Block Write 즉, Ctrl-KW를 사용하고 화일이름 대신에 LPT1 또는 PRN을 입력한다. PC

360KB 드라이브로 360KB이상의 데이터를 복사하려면

최명환

여기에 소개하는 프로그램은 640KB이상의 데이터를 연속적으로 복사하거나 복사받을 디스켓의 남은 용량을 체크하여 복사될 데이터보다 적을 경우 다른 디스켓으로 옮기는 기능도 가지고 있어 많은 양의 데이터를 관리해야 하는 작업에서 매우 편리하게 사용될 수 있을 것이다.

일반적으로 응용프로그램을 실행시킬 때, 특히 회계나 고정자산 등의 프로그램일 때는 많은 데이터화일들이 생기게 되는데 가끔씩은 360KB 이상의 화일이 생긴다. 심지어 1MB를 넘는 데이터화일도 있을 수 있으며 이같은 문제는 대량의 업무를 PC로 처리할 때 생기는 공통의 현상이다.

그런데 어떤 의도에서든지 이런 데이터화일을 다른 시스템의 하드디스크에 옮겨서 응용프로그램을 구동시키려고 할 때 난처한 입장에 처하게 된다.

다행이 1.2MB의 고밀도 플로피디스크 드라이브를 갖춘 경우는 1.2MB이하의 모든 데이터를 복사해서 옮길 수 있지만 360KB의 플로피디스크 드라이브를 갖추고 있는 경우는 위에서 언급한 360KB이상의 데이터를 복사하기가 곤란하다.

이런 문제에 대한 해결책으로는 하드웨어적으로 카세트테입(Cassette Tape)과 같은 백업장치를 추가로 설치하는 방법과 소스 데이터화일을 여러장의 디스켓에 분리시킨 후 하드디스크내에서 다시 결합하는 방법이 있을 것이다.

원래 첫번째 방법이 가장 이상적인 방법이라고 생각되지만 하드웨어를 구입하는데 적지않은 비용을 감수해야만 한다. 그래서 필자는 후자의 방법으로써 비용도 안들고 순전히 디스켓만 준비하면 복사할 수 있는 프로그램을 만들게 되었다(리스트 1 참조).

Cop.asm에는 640KB이상의 데이터나 프로그램을 연속적으로 복사하거나 복사받을 디스켓의 남은 용량을 체크하여 복사될 데이터보다 적을 경우 다른 디스켓으로 옮겨 복사하는 테크닉도 소개하고 있으므로 다른 용도로도 이 방법을 응용해 보기 바란다.

또한 미약하기는 하나 화면을 편집하는 루틴도 넣었다. 이 루틴은 윈도우내에서 상하좌우로 이동이 가능하며 삽입(Insert) 모드와 삭제(Delete)모드가 있고, 백 스페이스(Back Space)기능이 가능하다. 또 〈Enter〉키를 치면 보통의 에디터에서와 같이 한 줄을 스킵(Skip)하면서 커서가 화면의 좌측으로 위치하게 된다.

필자가 이와같은 프로그램을 처음 구상하게 된 동기는 노턴가이드의 Assembly Database를 본 후부터 이다.

노턴가이드를 잘아는 독자라면 Assembly Database가 600KB가 된다는 점과 하드디스크나 1.2MB의 고밀도 플로피디스크 드라이브가 없으면 이용할 수 없다는 것을 잘 알 것이다. 그런데 640KB의 RAM과 360KB의 드라이브 2대만 갖고 있는 필자로서는 이 노턴가이드를 이용할 수 없다는 사실이 무척 안타까왔다. 그러나 어떻게 해서든지 필자가 가지고 있는 컴퓨터 안에서 이 덩치큰 데이터베이스를 움직이게 하고 싶었다.(노턴가이드의 Database 내용을 어셈블리 언어 프로그래밍 작업에 이용하고 싶은 마음과 필자의 컴퓨터에서 실행시킬 수 없다는 현실에서 오기로 시작했으나 결국 프로그램을 실행시키고 보니 Database의 내용을 참조하는 것보다 그래도 이책 저책을 뒤적거리며 참조하는 것이 시각적으로 더 낫다는 사실을 알게 되었다.) 그래서 궁리하고 고민한 끝에 만들게 된 것이 Cop.asm이다.

Cop. EXE

Cop.asm을 컴파일해서 얻은 실행형 화일 Cop.EXE을 실행시키면 화면 1과 같이 윈도우가 나타난다.

좌측 윈도우의 SOURCE FILE NAME에는 복사하고자 하는 화일들이 들어가며 여기에 지정된 모든 화일들은 하나의 화일로 간주되어 TARGET로 보내진다. 우측 윈도우의 TARGET FILE NAME에는 복사될 화일 이름들이 들어가는데 SOURCE FILE의 데이터들은 TARGET FILE에 등

록된 이름으로 복사되면서 디스크의 남은 용량을 체크하여 부족하면 디스켓을 바꾸라는 메시지를 출력하고 대기상태가 된다. 이때 디스켓을 바꾸고 아무키나 누르면 복사가 계속 진행된다.

한편 이 프로그램 실행에 쓰이는 함수키는 표 1에 보인바와 같으며 F2와 F3는 내부적으로 동일한 루틴을 수행하고 있다. 이것은 사용자의 본래의 의도가 데이터를 분할할 것이냐 아니면 뭉칠 것이냐를 확실히 결정하고 조작하라는 의미에서 만든 것이다.

실행방법

한 예로 노턴가이드의 NG.NG의 경우를 보자. NG.NG는 600KB이므로 이 화일을 A드라이브의 디스켓에 D1, D2, D3로 분리하여 복사하는 경우이다.

일단 디스켓이 360KB의 빈디스켓이라면 D2까지 지정해도 괜찮지만 그 이하라면 D3까지 지정해 두는 것이 좋다(화면 2).

다음에는 함수키 〈F2〉를 누른다. 이미 설명했듯이 이 키는 화일을 분리시키는 기능을 수행하는데 컴퓨터는 D1화일에 300KB, D2화일에 300KB 정도의 비율로 연속해서 복사를 수행하게 된다.(D1으로 복사되는 도중 디스켓의 용량이 부족하면 디스켓을 바꾸라는 메시지를 출력하고, 새 디스켓을 넣어 임의의 키를 누르면 다음 행의 'a : d2' 화일로 그 여분의 데이터가 복사된다.)

참고로, 하드디스크에서 D1(300KB), D2(300KB)의 두 화일을 NG.NG라는 화일로 결합시키는 예를 화면 3에 나타내었다.

개선되어야 할 점

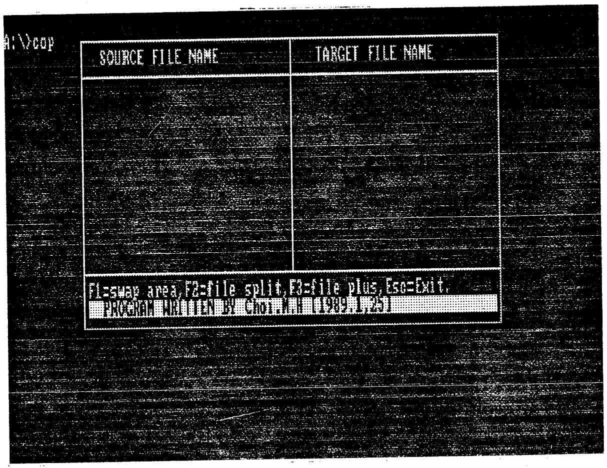
이 프로그램을 처음 만들었을 때 노턴가이드의 Database를 완전하게 이용할 수는 없었다. 처음에 ASM.NG(약 600KB)를 Cop.EXE를 이용하여 두개의 빈디스켓에 복사하고 (대략 300KB : 300KB의 비율로 되며 이미 디스켓에 다른 화일들이 있었다면 3장 이상으로 늘 수도 있다.) 그다음 DOS디스켓으로 램디스크 드라이브를 450~500KB로 잡은 후 2장의 디스켓에 있는 데이터를 램디스크안에서 다시 결합하였다. 물론 결합되는 도중에 디스켓을 바꾸라는 메시지가 나타나지만 램디스크이므로 할 수 없이 〈Ctrl〉-〈Break〉를 눌러 강제로 종료시켰다.

이렇게 되면 램디스크안에 대략 450~500KB의 ASM.NG 화일이 생기고, 100KB정도의 데이터 손실이 있게 되었다. 그나마 필요로 하는 정보는 웬만큼 얻을 수는 있었지만 이런 문제를 비롯하여, 프로그램을 램상주 형태로 만든 후 언제든지 핫키(Hot Key)만 누르면 윈도우가 나타나면서 디폴트 드라이브의 모든 데이터를 SOURCE에 표시하게 하거나 PC TOOLS의 COPY SELECT와 같이 TARGET NAME으로 복사하는 도중에 선택기능을 갖게 하는 등 여러기능을 추가시키면 더 좋을 개선점들이 많이 있다.

다음에 기회가 있으면 이 프로그램을 램상주로 만드는 방법을 소개하겠다.

이것은 가로채기상태에서 DOS기능호출을 실행하면 수행

〔화면 1〕 초기 실행화면



〔표 1〕 Cop.EXE에 쓰이는 함수키

키	기능
〈F1〉	커서를 SOURCE와 TARGET영역으로 이동시킨다.
〈F2〉	데이터를 분할하는 역할을 한다.
〈F3〉	데이터를 하나로 뭉치는 역할을 한다.
〈Esc〉	프로그램에서 빠져 나온다.
〈Ins〉	Insert 모드 전환
〈Del〉	커서의 현재 위치에서 한 문자를 지운다.
〈←〉	백 스페이스 기능
〈Enter〉	커서를 한줄 스킵하면서 맨 좌측으로 이동시킨다.

〔화면 2〕 화일을 분리하여 복사하는 경우

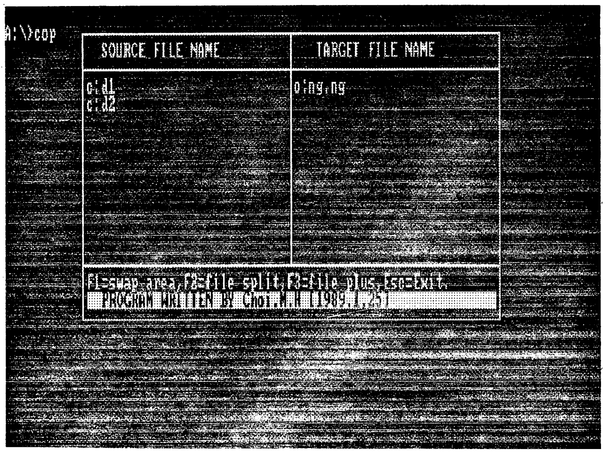
SOURCE FILE NAME	TARGET FILE NAME
c:ng.ng	a:d1 a:d2 a:d3] C드라이브의 데이터 화일을 D1, D2, D3로 분리시켜 복사한다.

```
F1=swap area,F2=file split,F3=file plus,Esc=Exit.
  PROGRAM WRITTEN BY Choi.M.H [1989.1,25]
```

이 잘 안되는 경우가 있기 때문에 이에 대한 해결책을 제시하고 싶기 때문이다.

끝으로 필자외에도 좀더 실력있는 독자의 참여를 기대해 본다. PC

〔화면 3〕 화일을 결합시키면서 복사하는 경우



〔리스트 1〕 Cop.asm

```
stack   segment  stack 'stack'
        db       21  dup(0)
stack   ends
code  segment  'code'
                assume   cs:code,ds:code,ss:stack
entry           proc     far
                mov      ax,cs
                mov      ds,ax
                mov      es,ax

                call     screen_edit

                mov      ah,4ch
                int      21h     TARGET 드라이브의 잔여용량을
disk space      proc  <───── 체크하여 복사하고자 하는 SOURCE
                                 데이터보다 양이 적으면 레이블
                mov      ah,19h   〔dsk-space 1〕에 'n'을,
                int      21h        많을 때는 'y'를 넣는다.
                mov      byte ptr [cur_drive],al
                lea      bx,path1
                cmp      byte ptr [bx+1],':'
                jnz      dsks_7
                mov      al,[bx]
                cmp      al,'a'
                jb       dsks_5
                sub      al,'a'
                jmp      dsks_6
dsks_5:         sub      al,'A'
dsks_6:         mov      ah,0eh
                mov      dl,al
                int      21h
dsks_7:
                mov      ah,36h
                mov      dl,00h
                int      21h
```

```
                mul      bx
                mul      cx
                mov      word ptr [dsk_space],dx
                mov      word ptr [dsk_space+2],ax
                cmp      word ptr [dsk_space],0000h
                jnz      dsk_5
                mov      ax,word ptr [file_length]
                cmp      word ptr [dsk_space+2],ax
                jnc      dsk_5
                mov      byte ptr [dsk_space1],'n'
                mov      ah,0eh
                mov      dl,byte ptr [cur_drive]
                int      21h
                call     dsk_space_error
                ret
dsk_5:          mov      byte ptr [dsk_space1],'y'
                mov      ah,0eh
                mov      dl,byte ptr [cur_drive]
                int      21h
                ret
dsk_space       dw       0,0
dsk_space1      db       'y'
cur_drive       db       00h
disk_space      endp
print_string    proc
                ret
                mov      ah,9
                int      21h
                ret
print_string    endp
prg_hi_seg      proc  <───── 프로그램 위의 빈 공간 번지를 얻는다.
                lea      ax,data3
                mov      cl,4
                shr      ax,cl
```

```
                inc     ax
                inc     ax
                mov     bx,cs
                add     ax,bx
                mov     word ptr [p_h_s],ax
                ret
prg_hi_seg      endp
printb          proc
                push    ax
                push    bx
                push    cx
                push    dx
                mov     ah,14        레지스터 AL의 내용을 프린트
                int     10h
                pop     dx
                pop     cx
                pop     bx
                pop     ax
                ret
printb          endp
print_hex       proc
                push    ax
                mov     cl,04h
                shr     al,cl
                cmp     al,09h
                jg      print_hex_1
                add     al,30h
                jmp     print_hex_2
print_hex_1:    add     al,37h
print_hex_2:    mov     dl,al
                call    printb       AL의 값을 ASCII로 변환
                pop     ax
                and     al,0fh
                cmp     al,09h
                jg      print_hex_3
                add     al,30h
                jmp     print_hex_4
print_hex_3:    add     al,37h
print_hex_4:    mov     dl,al
                call    printb
                ret
print_hex       endp
print_ax        proc
                push    ax
                mov     al,ah
                call    print_hex    AX의 값을 ASCII로 변환
                pop     ax
                call    print_hex
                ret
print_ax        endp
file_create     proc
                mov     ah,3ch ;file create    화일이름을
                lea     dx,path                만든다.
```

```
                mov     cx,20h
                int     21h
                mov     word ptr [file_handle],ax
                ret
file_create     endp
file_open       proc
                mov     ah,3dh ;file open      화일을 읽고
                mov     al,2 ;read or write    쓰기 위해서
                lea     dx,path                OPEN한다.
                int     21h
                mov     word ptr [file_handle],ax
                ret
f_o:            db      'file opening:$$'
file_open       endp                           화일이름을 만든다.
file_create1    proc
                mov     ah,3ch ;file create
                lea     dx,path1
                mov     cx,20h
                int     21h
                mov     word ptr [file_handle],ax
                ret
file_create1    endp                           화일을 읽고 쓰기 위해 OPEN
file_open1      proc
                mov     ah,3dh ;file open
                mov     al,2 ;read or write
                lea     dx,path1
                int     21h
                jc      fol_5
                mov     word ptr [file_handle],ax
                ret
fol_5:          call    file_create1
                ret
file_open1      endp                           OPEN한 화일의 화일포인터를 이동
file_pointer    proc
                mov     ah,42h ;moving file pointer
                mov     cx,word ptr [file_point]
                mov     dx,word ptr [file_point+2]
                mov     al,0 ;from first of file
                mov     bx,word ptr [file handle]
                int     21h
                ret                            Path 1으로
file_pointer    endp                           OPEN한 화일 끝으로 포인터를 이동
file_pointer1   proc    ;path1
                mov     ah,42h ;moving file pointer
                xor     cx,cx
                xor     dx,dx
                mov     al,2 ;from last of file
                mov     bx,word ptr [file_handle]
                int     21h
                ret
file_pointer1   endp                           OPEN시킨 화일을 닫는다.
file_close      proc
                mov     ah,3eh ;file close
```

```
                mov     bx,word ptr [file_handle]
                int     21h
                ret
file_close      endp                                화일 사이즈 계산
cac_file_size   proc
                mov     ah,42h ;moving file pointer
                xor     cx,cx
                xor     dx,dx
                mov     al,2 ;from last file
                mov     bx,word ptr [file_handle]
                int     21h ;dx:ax
                mov     word ptr [file_size],dx
                mov     word ptr [file_size+2],ax
                ret
cac_file_size   endp                                화일의 길이를 계산
cac_file_length proc
                cmp     word ptr [file_size],0000h
                jnz     cfl_5
                cmp     word ptr [file_size+2],0fff0h
                jnc     cfl_5
                mov     ax,word ptr [file_size+2]
                mov     word ptr [file_length],ax
                xor     ax,ax
                mov     word ptr [file_size+2],ax
                ret
cfl_5:  ;------- file_size(xxxx:0000) - 0fff0h ------
                mov     ax,word ptr [file_size+2]
                clc
                sub     ax,0fff0h
                sbb     word ptr [file_size],00h
                mov     word ptr [file_size+2],ax
                mov     word ptr [file_length],0fff0h
                ret
cac_file_length endp    OPEN시킨 화일로부터 데이터를 읽는다.
file_read       proc
                mov     ah,3fh
                mov     cx,word ptr [file_length]
                mov     bx,word ptr [file_handle]
                mov     dx,word ptr [p_h_s]
                mov     ds,dx
                mov     dx,00h
                int     21h
                mov     ax,cs
                mov     ds,ax
                ret
file_read       endp    OPEN시킨 화일로부터 데이터를 써넣는다.
file_write      proc
                mov     ah,40h ;file write
                mov     cx,word ptr [file_length]
                mov     bx,word ptr [file_handle]
                mov     dx,word ptr [p_h_s]
                mov     ds,dx
                mov     dx,00h
                int     21h
                mov     ax,cs
                mov     ds,ax
                ret
file_size       dw      0,0
file_length     dw      0
file_point      dw      0,0 ;of path
file_point1     dw      0,0 ;of path1
file_write      endp
path_second     proc            SOURCE의 Path를 읽는다.
                mov     al,byte ptr [sxy_no]
                mov     byte ptr [input_no1],al
                mov     ax,word ptr [sxy1]
                mov     word ptr [sxy2],ax
                lea     di,path
                call    prmt_to_ptah
                cmp     ax,'er'
                jnz     pth_se_5
                push    ax
                ;call   source_error
                pop     ax
                ;mov    ah,4ch
                ;int    21h
pth_se_5:       ret
sxy_no          db      00h
path_second     endp                    TARGET의 Path를 읽는다.
path1_second    proc
                cmp     byte ptr [skip_input_no1],'y'
                jnz     p_s_5
                inc     byte ptr [txy_no]
                mov     byte ptr [skip_input_no1],'n'
p_s_5:          ;
                mov     al,byte ptr [txy_no]
                mov     byte ptr [input_no1],al
                mov     ax,word ptr [txy1]
                mov     word ptr [sxy2],ax
                lea     di,path1
                call    prmt_to_ptah
                cmp     ax,'er'
                jnz     pt_se_5
                call    target_error
                mov     ah,4ch
                int     21h
pt_se_5:        ret
txy_no          db      00h
skip_input_no1  db      'n'
path1_second    endp                    SOURCE 화일의
add_point       proc                    오프셋 데이터 포인터를 구한다.
                mov     ax,word ptr [file_length]
                clc
                add     word ptr [file_point+2],ax
                adc     word ptr [file_point],0000h
                ret
```

```
add_point       endp
add_point1      proc
                mov     ax,word ptr [file_length]
                clc
                add     word ptr [file_point1+2],ax
                adc     word ptr [file_point1],0000h
                ret
add_point1      endp
file_proc       proc ──────화일을 복사하는 루틴
                mov     ax,cs
                mov     ds,ax
                mov     byte ptr [skip_input_no1],'n'
f_p_4:          mov     word ptr [file_point],00h
                mov     word ptr [file_point+2],00h
                call    file_open
                jc      error
                call    cac_file_size
                jc      error
f_p_5:
                call    file_close
                call    file_open
                jc      error
                call    cac_file_length
                jc      error
                call    file_pointer
                jc      error
                call    file_read
                jc      error
                call    file_close
                jc      error
fp_5:           call    disk_space
                cmp     byte ptr [dsk_space1],'y'
                jz      f_p_5_5
                mov     ah,0
                int     16h
                call    blank_error
                call    path1_second
                jmp     fp_5
f_p_5_5:        call    file_open1      ;path1
                jc      error
                call    file_pointer1   ;last
                jc      error
                call    file_write
                jc      error
                mov     byte ptr [skip_input_no1],'y'
                cmp     word ptr [file_size],0000h
                jnz     f_p_6
                cmp     word ptr [file_size+2],0000h
                jz      f_p_7
f_p_6:          ;
                call    add_point
                call    file_close
                jmp     f_p_5
f_p_7:          ;
                call    file_close
                jc      error
                inc     byte ptr [sxy_no]
                call    path_second
                cmp     ax,'er'
                jz      f_p_8
                jmp     f_p_4
f_p_8:          ;
                ret          디스크 에러가 발생하면
                             에러 메시지와 에러 번호를 출력
error:
                call    file_close
                call    error_blink
                mov     ah,2
                mov     bh,0
                mov     dx,word ptr [error_po]
                int     10h
                lea     dx,err_msg
                mov     ah,9
                int     21h
                mov     ax,word ptr [file_handle]
                call    print_ax
                ret
prt_err:        mov     ah,4ch
                int     21h
err_msg         db      'Disk error --->$'
file_proc       endp
screen_write            proc ──────화면의 Map을 표시
                mov     ax,word ptr [xy1]
                mov     word ptr [xy2],ax
                mov     ah,2
                mov     bh,0
                mov     dx,word ptr [xy2]
                int     10h
                mov     ah,9
                lea     dx,dis1
                int     21h
                inc     byte ptr [xy2+1]
                mov     ah,2
                mov     bh,0
                mov     dx,word ptr [xy2]
                int     10h
                mov     ah,9
                lea     dx,dis2
                int     21h
                inc     byte ptr [xy2+1]
                mov     ah,2
                mov     bh,0
                mov     dx,word ptr [xy2]
                int     10h
                mov     ah,9
                lea     dx,dis3
                int     21h
```

```
            inc     byte ptr [xy2+1]
            mov     cx,10
dspy_5:     push    cx
            mov     ah,2
            mov     bh,0
            mov     dx,word ptr [xy2]
            int     10h
            mov     ah,9
            lea     dx,dis4
            int     21h
            inc     byte ptr [xy2+1]
            pop     cx
            loop    dspy_5
            mov     ah,2
            mov     bh,0
            mov     dx,word ptr [xy2]
            int     10h
            mov     ah,9
            lea     dx,dis5
            int     21h
            inc     byte ptr [xy2+1]
            mov     ah,2
            mov     bh,0
            mov     dx,word ptr [xy2]
            int     10h
            mov     ah,9
            lea     dx,dis6
            int     21h
            inc     byte ptr [xy2+1]

            mov     ah,2
            mov     bh,0
            mov     dx,word ptr [xy2]
            inc     dl
            mov     word ptr [error_po],dx
            int     10h
            mov     ah,9 ;write at cur.
            mov     al,' '
            mov     bh,0
            mov     cx,55
            mov     bl,112
            int     10h
            mov     ah,2
            mov     bh,0
            mov     dx,word ptr [xy2]
            int     10h
            mov     ah,9
            lea     dx,dis7
            int     21h
            inc     byte ptr [xy2+1]
            mov     ah,2
            mov     bh,0
            mov     dx,word ptr [xy2]
```

```
                int     10h
                mov     ah,9
                lea     dx,dis8
                int     21h
                inc     byte ptr [xy2+1]
                ret
xy1             db      0ah,01h ;X,Y
xy2             dw      00h
dis1     db '┌────────────────────────────┬────────────────────────────┐$'
dis2     db '│ SOURCE FILE NAME           │      TARGET FILE NAME      │$'
dis3     db '├────────────────────────────┼────────────────────────────┤$'
dis4     db '│                            │                            │$'
dis5     db '├────────────────────────────┴────────────────────────────┤$'
dis6     db '│F1=swap area,F2=file split,F3=file plus,Esc=Exit.        │$'
dis7     db '│ PROGRAM WRITTEN BY Choi.M.H [1989.1.25]                 │$'
dis8     db '└─────────────────────────────────────────────────────────┘$'
screen_write            endp
set_locate      proc
                mov     dx,word ptr [sxy2]
                add     dl,byte ptr [input_len1] ;X
                dec     dl
                add     dh,byte ptr [input_no1] ;Y
                dec     dh
                mov     ah,2
                mov     bh,0
                int     10h
                ret
```

커서 위치를 세트

```
set_locate      endp
right           proc
                mov     al,byte ptr [input_len1]
                cmp     byte ptr [input_len],al
                jz      rt_5
                inc     byte ptr [input_len1]
                jmp     rt_6
                ret
rt_5:           mov     al,byte ptr [input_no1]
                cmp     byte ptr [input_no],al
                jz      rt_6
                inc     byte ptr [input_no1]
                mov     byte ptr [input_len1],01h
rt_6:           ;
                call    set_locate
                ret
```

커서를 오른쪽으로 한칸 이동

```
right           endp
left            proc
                mov     al,byte ptr [input_len1]
                cmp     al,1
                jz      lt_5
                dec     byte ptr [input_len1]
                jmp     lt_6
                ret
lt_5:           mov     al,byte ptr [input_no1]
                cmp     al,1
```

커서를 왼쪽으로 한칸 이동

```
        jz      lt_6
        dec     byte ptr [input_no1]
        mov     dl,byte ptr [input_len]
        mov     byte ptr [input_len1],dl
lt_6:   call    set_locate
        ret
left    endp
up      proc                              ; 커서를 위로 이동
        cmp     byte ptr [input_no1],1
        jz      up_5
        dec     byte ptr [input_no1]
up_5:   call    set_locate
        ret
up      endp
down    proc                              ; 커서를 밑으로 이동
        mov     dh,byte ptr [input_no1]
        cmp     byte ptr [input_no],dh
        jz      down_5
        inc     byte ptr [input_no1]
down_5: call    set_locate
        ret
down    endp
delete  proc    ; 커서 위치의 한 문자를 지우고 다음에 이어지는 한 행의 모든 문자를 한칸 왼쪽으로 당긴다.
        mov     byte ptr [del_data1],' '
        mov     al,byte ptr [input_len1]
        mov     byte ptr [del_save],al
        mov     al,byte ptr [input_len]
        mov     byte ptr [input_len1],al
        call    set_locate
del_5:  mov     ah,8
        mov     bh,0
        int     10h
        mov     byte ptr [del_data],al
        mov     al,byte ptr [del_data1]
        mov     ah,10
        mov     bh,0
        mov     cx,1
        mov     bl,7
        int     10h
        mov     dl,byte ptr [del_save]
        cmp     byte ptr [input_len1],dl
        jz      del_6
        call    left
        mov     al,byte ptr [del_data]
        mov     byte ptr [del_data1],al
        jmp     del_5
del_6:  ret
del_save db     00h
del_data db     00h
del_data1 db    00h
delete  endp
insert  proc    ; 키보드로부터 입력받은 한 문자를 삽입
        mov     byte ptr [ins_data1],al
        mov     al,byte ptr [input_len1]
        mov     byte ptr [ins_save],al
        call    set_locate
ins_5:  mov     ah,8
        mov     bh,0
        int     10h
        mov     byte ptr [ins_data],al
        mov     al,byte ptr [ins_data1]
        mov     ah,10
        mov     bh,0
        mov     cx,1
        mov     bl,7
        int     10h
        mov     dl,byte ptr [input_len1]
        cmp     byte ptr [input_len],dl
        jz      ins_6
        call    right
        mov     al,byte ptr [ins_data]
        mov     byte ptr [ins_data1],al
        jmp     ins_5
ins_6:  ;
        mov     dl,byte ptr [ins_save]
        mov     byte ptr [input_len1],dl
        call    set_locate
        call    right
        ret
ins_save db     00h
ins_data db     00h
ins_data1 db    00h
insert  endp
trans_insert proc   ; 현재의 상태가 삽입모드이면 해제하고, 아니면 삽입모드로 세트
        cmp     byte ptr [t_i],'y'
        jz      t_i_5
        mov     byte ptr [t_i],'y'
        mov     ah,01h ;insert mode
        mov     ch,3
        mov     cl,13
        int     10h
        ret
t_i_5:  ;
        mov     byte ptr [t_i],'n'
        mov     ah,01h ;insert mode
        mov     ch,12
        mov     cl,13
        int     10h
        ret
t_i     db      'n'
trans_insert endp
screen_edit proc    ; 키보드로부터 입력된 데이터를 처리
        call    screen_write
        mov     dx,word ptr [xy1]
        add     dh,3
        inc     dl
```

```
            mov     word ptr [sxyl],dx ;source
            mov     word ptr [sxy2],dx
            add     dl,28
            mov     word ptr [txyl],dx ;target
            call    right
            call    left

s_e_5:      mov     ah,0
            int     16h
            cmp     ax,4b00h   → "→"처리
            jnz     s_e_6
            call    left
            jmp     s_e_5
s_e_6:      cmp     ax,4d00h   → "←"처리
            jnz     s_e_7
            call    right
            jmp     s_e_5
s_e_7:      ;
s_e_7_1:    cmp     ax,011bh
            jnz     s_e_8
            ret
            jmp     s_e_5
s_e_8:      cmp     ax,4800h   → "↑"처리
            jnz     s_e_9
            call    up
            jmp     s_e_5
s_e_9:      cmp     ax,5000h   → "↓"처리
            jnz     s_e_10
            call    down
            jmp     s_e_5
s_e_10:     cmp     ax,5300h
            jnz     s_e_10_1
            call    delete
            jmp     s_e_5
s_e_10_1:   cmp     ax,1c0dh
            jnz     s_e_10_2
            mov     byte ptr [input_len1],01h
            call    down
            jmp     s_e_5
s_e_10_2:   cmp     ax,3c00h ;if  F2
            jnz     s_e_10_3
            mov     byte ptr [input_no1],01h
            mov     byte ptr [sxy_no],01h
            mov     byte ptr [txy_no],01h
            call    f2_key
            jmp     s_e_5
s_e_10_3:   cmp     ax,3d00h ;if  F3
            jnz     s_e_11
            mov     byte ptr [input_no1],01h
            mov     byte ptr [sxy_no],01h
            mov     byte ptr [txy_no],01h
            call    f2_key
            jmp     s_e_5
```

```
s_e_11:         cmp     ax,5200h
                jnz     s_e_11_1
                call    trans_insert
                jmp     s_e_5
s_e_11_1:       cmp     ax,0e08h
                jnz     s_e_11_2
                call    left
                call    delete
                jmp     s_e_5   커서 위치의 영역을 바꾼다.
s_e_11_2:       cmp     ax,3b00h
                jnz     s_e_12
                ;--------- swap area --------
                mov     dx,word ptr [sxy2]
                cmp     word ptr [sxyl],dx
                jnz     s_e_11_3
                mov     dx,word ptr [txyl]
                mov     word ptr [sxy2],dx
                mov     byte ptr [input_len1],01h
                call    right
                call    left
                jmp     s_e_5
        s_e_11_3:;
                mov     dx,word ptr [sxyl]
                mov     word ptr [sxy2],dx
                mov     byte ptr [input_len1],01h
                call    right
                call    left
                jmp     s_e_5
s_e_12:         ;write at current
                cmp     al,32
                jge     s_e_12_4
                jmp     s_e_5
s_e_12_4:       cmp     byte ptr [t_i],'y'
                jnz     s_e_12_5            삽입모드로 세트
                call    insert
                jmp     s_e_5
s_e_12_5:
                mov     ah,10
                mov     bh,0
                mov     cx,1
                mov     bl,7
                int     10h
                call    right
                jmp     s_e_5
                ret
input_len       db      25
input_no        db      10
input_len1      db      1
input_no1       db      1
sxyl            dw      00h
sxy2            dw      00h
txyl            dw      00h
screen_edit     endp
```

```
prmt_to_ptah    proc    ; 윈도우 안의 TARGET영역과 SOURCE영역으로부터 Path이름을 전송
                mov     al,byte ptr [input_len]
                inc     al
                inc     al
                mov     byte ptr [input_len1],al
                call    set_locate
                mov     al,byte ptr [input_no1]
                add     al,30h
                mov     ah,9
                mov     bh,0
                mov     bl,0f7h
                mov     cx,1
                int     10h
                mov     byte ptr [input_len1],01h
                call    set_locate
                mov     al,byte ptr [input_no1]
                mov     byte ptr [ip_no],al
                mov     al,byte ptr [input_len]
                mov     byte ptr [ip_len],al
                mov     ax,cs
                mov     es,ax
ptp_5:          mov     ah,8 ;read at cur.
                mov     bh,0
                int     10h
                cmp     al,'!'
                jge     ptp_6
                call    right
                dec     byte ptr [ip_len]
                jz      ptp_error
                jmp     ptp_5
ptp_6:          mov     ah,8 ;read at cur.
                mov     bh,0
                int     10h
                mov     es:[di],al
                mov     ah,9 ;write at cur.
                mov     bh,0
                mov     cx,1
                mov     bl,112
                int     10h
                call    right
                inc     di
                dec     byte ptr [ip_len]
                jz      ptp_7
                jmp     ptp_6
ptp_7:          ;
                mov     es:byte ptr [di],00h
                inc     di
                mov     es:byte ptr [di],'$'
                mov     al,byte ptr [ip_no]
                mov     byte ptr [input_no1],al
                mov     ax,00h
                ret
ptp_error:
                mov     al,byte ptr [ip_no]
                mov     byte ptr [input_no1],al
                mov     ax,'er'
                ret
ip_len          db      00h
ip_no           db      00h
prmt_to_ptah    endp
error_blink     proc    ; 에러 표시의 속성을 깜박거리게 바꾼다.

                mov     ah,2
                mov     bh,0
                mov     dx,word ptr [error_po]
                int     10h
                mov     ah,9 ;write at cur.
                mov     al,' '
                mov     bh,0
                mov     cx,55
                mov     bl,0f0h
                int     10h
                ret
error_blink     endp
error_normal    proc    ; 에러 표시의 속성을 정상으로 바꾼다.

                mov     ah,2
                mov     bh,0
                mov     dx,word ptr [error_po]
                int     10h
                mov     ah,9 ;write at cur.
                mov     al,' '
                mov     bh,0
                mov     cx,55
                mov     bl,07h
                int     10h
                ret
error_normal    endp
source_error    proc
                call    error_blink
                mov     ah,2
                mov     bh,0
                mov     dx,word ptr [error_po]
                dec     dl
                int     10h
                lea     dx,err1
                mov     ah,9
                int     21h
                ret
error_po        dw      00h
err1    db '|  SOURCE FILE NAME PATH NOT FOUNDED.|$'
source_error    endp
blank_error     proc
                call    error_normal
                mov     ah,2
                mov     bh,0
```

```
                mov     dx,word ptr [error_po]
                dec     dl
                int     10h
                lea     dx,err4
                mov     ah,9
                int     21h
                ret
err4    db '|                                       ,$|
blank_error     endp
target_error    proc
                call    error_blink
                mov     ah,2
                mov     bh,0
                mov     dx,word ptr [error_po]
                dec     dl
                int     10h
                lea     dx,err2
                mov     ah,9
                int     21h
                ret
err2    db '|  TARGET FILE NAME.PATH NOT FOUNDED.,$|
target_error    endp
dsk_space_error proc
                call    error_blink
                mov     ah,2
                mov     bh,0
                mov     dx,word ptr [error_po]
                dec     dl
                int     10h
                lea     dx,err3
                mov     ah,9
                int     21h
                ret
err3    db '|NOT ENOUGH DISK SPACE.
            CHANGE DISK ANOTHER WILL COPY.  |$'
dsk_space_error endp
f2_key          proc
                mov     ax,word ptr [sxy1]
                mov     word ptr [sxy2],ax
                lea     di,path
                call    prmt_to_ptah
                cmp     ax,'er'
                jnz     F2_5
                call    source_error
                mov     ah,4ch
                int     21h
F2_5:           ;
                lea     dx,path
                call    print_string
F2_5_5:         mov     ax,word ptr [txy1]
                mov     word ptr [sxy2],ax
                lea     di,path1
                call    prmt_to_ptah
                cmp     ax,'er'
                jnz     F2_6
                call    target_error
                mov     ah,4ch
                int     21h
F2_6:           ;
                lea     dx,path1
                call    print_string
                call    prg_hi_seg
                call    file_proc
                mov     ah,4ch
                int     21h
f2 key          endp
path            db 80 dup (' ')   ┐ 키보드로부터 입력받은
path1           db 80 dup (' ')   ┘ 파라미터
                ;db 2  dup(0)
data1           db 50 dup (' ')
file_handle     dw 00h                  프로그램의 최상위 번지
p_h_s           dw 0,0 ;prg_hi_seg ─┘
data2           db 100h dup (' ')  ┐ 프로그램 코드와 복사
data3           db ' '             │ 데이터의 충돌을 방지하기
entry           endp               ┘ 위해서 버퍼영역을 만든다.
;-----------------------------------
                code            ends
                end             entry
```

국내소식

- 과학기술처
국민연금관리공단에 IBM기종 도입 승인

과기처는 올해 첫 컴퓨터도입 조정소위원회를 열고 국민연금관리공단이 신청한 IBM4381 모델 1대와 디스크 드라이브 프린터 등 컴퓨터 도입 계획을 승인했다.

국민연금관리공단은 연금업무 전산화를 위해 지난해 행정전산망용 주컴퓨터로 선정되어 있는 톨러런트시스템 8대를 22억 3천만원에 구입하여 가동했으나 성능취약, 시스템의 불안정 등으로 장애를 일으켜 새로운 컴퓨터도입을 검토해 왔었다.

연금관리공단측은 IBM 제품이 연금업무에 적합하다고 판단 이의 도입을 추진했으나 IBM컴퓨터의 통신방식이 국제표준과 달라 국가기간전산망에 연결문제가 대두됨으로써 결정이 보류되었다. 그러나 컴퓨터도입조정 위원회는 IBM이 91년까지 국제표준컴퓨터 통신방식인 OSI에 공급기종을 연결해 주는 조건으로 도입계획을 승인했다.

한편 이번 컴퓨터도입 조정위원회에서는 모두 12대의 도입계획이 승인되었다.

- 기계연구소 대덕선박분소
집중모니터링 시스템 개발

선박엔진의 운전상태를 원격감시, 제어할 수 있는 집중모니터링 시스템이 개발됐다.

이에 따라 외국기술에 의존해 오던 국내선박 자동화분야에 활력소가 될 것으로 기대되고 있다.

기계연구소의 대덕선박분소 김병덕 연구팀은 민간업체인 KT전기(주)와 2년간의 공동연구끝에 선박디젤기관 집중모니터링 시스템을 개발하는데 성공했다.

이 모니터링 시스템은 선박주기관과 격리되어 있는 기관 제어실에서 기관의 운전상태 요소를 마이크로컴퓨터를 이용해 감시제어하도록 되어 있다.

이 시스템은 엔진운전 상태를 나타내는 온도, 압력레벨, 회전수 등 각 물리량을 전기신호로 변환하는 센서부, 센서의 출력신호를 디지탈신호로 바꾸는 신호변환부, 디지탈신호를 처리하는 마이크로프로세서부, 처리신호를 감시자에게 알려주는 디스플레이부, 엔진관련 기기를 원격으로 자동제어할 수 있는 신호출력부로 구성되어 있다. 특히 이 시스템은 진동, 온도, 습도 등 선박의 열악한 환경에서도 견딜 수 있어 충분한 수출경쟁력을 가지고 있는 것으로 평가되고 있다.

- 상공부
POS기기 국산화 기술 개발 지원키로

상공부는 유통근대화의 필요성이 날로 증대됨에 따라 유통업체들의 POS(판매시점관리) 시스템도입에 따른 세제상지원 및 POS기기의 국산화를 위한 기술개발지원 등 POS 시스템도입 확대정책을 적극 추진하기로 했다.

상공부는 지난해 POS 시설도입업체에 대해 설비투자금액 10%의 세액을 공제지원한데 이어 최근 POS도입업체에 대한 소득표준 인하를 위해 재무부와 심의를 벌이고 있는 것으로 전해졌다.

그리고 협의가 끝나는 대로 기본율에 일정율을 감한 차등률을 적용하여 실시할 방침이다. 또한 연 15%의 수입 POS기기에 대한 관세를 내년13%, 91년11% 등으로 점차 인하해 나갈 계획이다.

이와함께 올해 책정된 150억원의 유통근대화자금의 일부를 중소유통업체의 POS기기 구입자금으로 대여할 계획도 세워놓고 있다.

그리고 상공부는 POS기기의 국산화를 위해 공업기반 기술개발과제로 선정하여 지난해 7천5백만원의 개발자금을 지원했으며 이중 한국컴퓨터 기술연구소가 추진중인 국산 POS 터미널 개발연구는 올해안으로 하고 과학기술원이 주관하고 있는 한국형POS 시스템개발은 오는 90년에 완료될 것으로 알려져 값싼 POS기기 공급에 따른 POS 도입은 더욱 확대될 것으로 전망된다.

- 내무부
행정망용 워크스테이션 성능시험 S/W 개발

컴퓨터를 이용하는 행정전산망용 워크스테이션의 각종기능 및 성능을 정확하게 테스트할 수 있는 프로그램이

내무부 주민관리전산화 전담 추진반에 의해 개발되어 관심을 모으고 있다.

추진반 신상칠씨는 최근 행정전산망용 워크스테이션의 CPU를 비롯하여 모니터, 프린터, 키보드, 디스크드라이브 등 관련 주변장치의 속성이나 기능 상태를 명확하게 측정 평가할 수 있는 워크스테이션 기능시험용 프로그램을 개발하였다.

이에 따라 종래 행정전산망용 워크스테이션 선정과정에서 발생했던 오류를 방지하여 객관적이고 합리적인 방법을 통해 적정기종을 선정할 수 있을 뿐 아니라 기종 선정일정을 단축하여 업무의 효율성을 기대할 수 있다.

C프로그래밍 언어로 작성된 이 소프트웨어는 기본 사칙연산 프로그램을 내장하여 컴퓨터의 처리결과를 정확하게 디스플레이할 수 있으며 디스크 드라이브내에서 데이터의 읽기 쓰기를 여러형태로 분리, 점검할 수 있도록 설계되어 있다.

또한 모니터의 컬러 및 흑백지원 능력은 물론 모니터의 해상도까지 상세하게 체크할 수 있을 뿐아니라 키스위치의 각종 기능이 프로그래밍되어 있어 컴퓨터화면을 통해 키보드의 기능과 고장 유무를 일일이 점검할 수도 있다.

이 소프트웨어는 워크스테이션 선정일정을 효율적으로 관리할 수 있는 일자, 시간 시험기능도 포함되어 있다.

한편 내무부는 앞으로 이 소프트웨어를 행정전산망용 워크스테이션 적부심사에 적극활용할 방침이다.

-시스템공학센터
크레이 수퍼컴퓨터 일반 서비스 실시 예정

시스템공학센터는 그동안 전용회선을 통해서만 접속을 허용해온 수퍼컴퓨터의 이용 효율을 높이기 위하여 일반교환회선(PSTN)을 통한 다이얼업 방식의 접속도 허용하기로 했다.

이를 위하여 시스템공학센터는 최근 통신공사와 계약을 맺고 회선구성방법 및 전화번호선정 등에 관한 세부적인 사항을 협의중인 것으로 알려졌다.

1차로 일반교환회선 10회선을 통해 시스템공학센터가 보유중인 "크레이 2S" 수퍼컴퓨터와 CDC, NAS, IBM 등 대형컴퓨터의 컴퓨터파워 및 센터가 소장하고 있는 수학통계, 그래픽스, 인공지능, 각종언어 등 16개 분야의 각종 응용소프트웨어 패키지서비스를 제공할 방침이다.

시스템공학센터는 이용량에 따라 서비스회선수를 늘려나가는 한편 지방이용자의 통신료 부담을 줄이기 위하여 통신공사의 협조를 얻어 시내통화료만으로 이같은 서비스를 이용할 수 있도록 특수번호 도입을 검토중이다. 시스템공학 센터는 이용자의 편의를 위해 1.2K에서부터 9.6Kbps까지의 다양한 속도를 제공할 방침이다.

-체신부
전산망 표준화작업 본격 추진

서로 다른 전산망과 이기종 컴퓨터간의 원활한 정보교환을 위한 전산망 표준화작업이 올해부터 본격 추진된다.

정보화시대에 대비하여 전산망기술 표준화추진 계획을 마련하고 한국전산원, 정보통신진흥협회, 전기통신기술협회 등 민간관련기관의 주도로 표준화 추진체제를 구축키로 했다.

추진일정은 올해부터 오는 99년까지 3단계로 나누어 실시될 예정인데, 체신부는 1단계인 89~91년에는 추진체제 구축과 함께 표준화기술의 자체개발, 민간주도에 의한 추진, 표준화대상을 범용성과 파급효과가 큰 분야부터 선택, 표준화 내용은 국제표준화 기구인 OSI를 근간으로 한다는 기본표준을 제정할 예정이다.

체신부는 이기간에 통신규약 시험지원센터의 설립과 개방형 통신규약표준안 작성 및 퍼스널컴퓨터통신처리 서비스인 MHS기능 표준안작성도 함께 마무리짓기로 했다.

2단계인 92~95년에는 전산망기술 표준화의 확립과 응용에 주안점을 두어 서비스별 표준제정확대에 주력할 방침이다.

3단계인 96~99년에는 전산망기술과 ISDN(종합정보통신망)과의 연계보완 부분을 집중연구하여 정착시킬 계획이다.

체신부는 전산망기술 표준화작업에 고도의 전문성이 필요한 점을 감안하여 연구와 기술개발을 한국전자통신연구소 주관으로 추진토록 할 구상이다.

-체신부
정보통신 역무제공업체 설립조건 완화

체신부는 정보통신 이용자가 급증하는 것과 관련하여 현재 사전승인제로 묶여있는 정보통신 역무제공업체 설립조건을 완화해 순수 자체 이용의 경우 신고제로 바꾸고 이용수수료를 받는 사업업체만 현형태로 승인제를 유지키로 했다.

체신부는 정보이용 서비스의 확산을 위해 정보통신 역무제공업체에 대한 설립절차를 이원화하여 기업내부 또는 계열기업간에만 이용하는 경우에는 신고만으로 설립을 할 수 있도록 할 계획이다.

반면 서비스 수수료를 받는 사업체의 경우 이용질서의 공정성 등의 문제점을 감안하여 현재대로 사전승인제를 적용키로 했다. 체신부는 다만 이같은 방침을 현행 공중전기통신사업법과 시행령을 개정하지 않고 내부조치로만 사용할 것으로 알려졌다.

-총무처
워드프로세서 구입 다양화키로

총무처는 행정전산망용 워드프로세서의 품질향상과 제품 다양화를 위해

기존사용중인 워드프로세서와 데이터 공유가 가능한 워드프로세서의 구입을 확대해 나가기로 했다.

총무처는 그동안 금성소프트웨어의 하나워드프로세서만 구입하던 독점구매 방식을 전면 수정하여 기존 하나워드프로세서와 100% 데이터호환성을 갖춘 워드프로세서를 행정전산망용 워드프로세서로 추가 선정키로 했다. 이를 위해 최근 행정전산망용 호환워드프로세서의 추가도입 계획을 확정하여 발표하는 한편 워드프로세서 호환성 평가항목을 마련하여 업계의 의견 수렴에 적극나서고 있다.

또한 이를 바탕으로 한국전산원과 공동으로 전산망에 관한 기술표준화를 검토하여 워드프로세서의 표준호환성 항목을 확정 공표한후 적당한 워드프로세서를 선정하여 각 정보기관에 구입추천을 의뢰할 방침이다.

-한국전기통신공사

G4 팩시밀리 상용서비스키로

한국전기통신공사는 고속전송시스템에 대한 수요증대에 부응하기 위해 올해 하반기부터 국산 중소용량 전자교환기인 TDX1A와 TDX1B를 이용하여 현재의 G3 팩시밀리보다 전송속도가 23~46배나 빠르면서도 선명도가 인쇄수준을 능가하는 G4팩시밀리를 상용 서비스하기로 했다. 이를 위하여 전기통신공사는 자체기술로 TDX1A와 TDX1B 교환기를 통해 데이터서비스를 할 수 있는 중계장치인 DSLB를 최근 개발한데 이어 14대의 G4 팩시밀리를 수입하여 전국 14개소의 지사 및 직할관서를 연결한 시범운용도 끝마쳤다.

한대당 가격이 1700만~1800만원선인 G4팩시밀리는 전화망을 포함한 공중통신망을 이용하게 되는데 전송속도는 56Kbps로 기존 G3 팩시밀리의 전송속도보다 최대 46배나 빠르다.

56Kbps속도는 1초에 알파벳문자의 경우 7천자, 한글은 2천자를 전송할 수 있는 성능이다. 또한 G4 팩시밀리는 전송선명도가 뛰어나 정밀설계도와 제판필름도 전송이 가능하다.

-한국과학기술원

시각 판별 조립장치 개발

시각장치와 로보트를 이용해 표면실장기술(SMT)을 실현할 수 있는 자동조립시스템이 과학기술원 변증남박사팀에 의해 개발됐다.

과기처 특정연구과제로 지난 3년간 수행해온 자동조립시스템은 시각장치로 조립부위를 판별한뒤 로보트에 의해 전자부품을 자동조립하는 첨단 생산자동화시스템이다.

이 시스템의 주요 요소인 인공시각장치와 패턴인식법, 고속정밀 서보시스템, 로보트 조정장치를 포함한 관리시스템 등이 모두 개발되었다.

이 시스템은 각종 생산가공 공정은 물론 각종 미세 조립공정까지 수행할 수 있다.

-한국전기통신공사

MSP 상반기중 상용화

한국전기통신공사는 올상반기중에 국제메시지 교환전용 통신서비스(MPS)를 상용화하여 지사 상호간에 메시지 전송과 퍼스컴, 팩시밀리 등의 통신이 가능하도록 할 계획이다.

지금까지는 전용회선을 가지고 있는 기업일 경우라도 국내본사와 미국지사간, 국내본사와 일본지사간에 전용회선이 구성되어 있는 구간에서만 통신이 가능했으나 연내에 MPS를 통해 지사끼리도 통신을 할 수 있게 된다.

또 MPS가 실용화되면 현재의 전용회선으로는 팩시밀리를 통해 전송할때 1분에 A_4용지 한장분량의 정보만 보낼 수 있으나, 고속부호급 전용회선(IBS)이 실용화되면 10장정도를 송신할 수 있다.

-동양정밀

HA사업 본격 참여할 예정

동양정밀(대표 : 박태선)이 가정자동화(HA) 사업에 본격 참여한다.

동양정밀은 최근 아파트, 빌라, 주택등에 설치해 각종 재해를 예방할 수 있는 가정 안전관리시스템을 자체기술로 개발했다.

이번에 개발한 이 가정 안전관리시스템은 내방객의 모습을 확인하고 대화할 수 있는 비디오 영상기능을 갖춘 비디오인터폰과 각종시설을 원격조정할 수 있는 휴대용 리모콘 등으로 구성되어 있다. 동양정밀은 이 시스템을 수출키 위해 현재 미국 및 유럽지역의 바이어와 수출상담을 진행중에 있는 것으로 알려졌다.

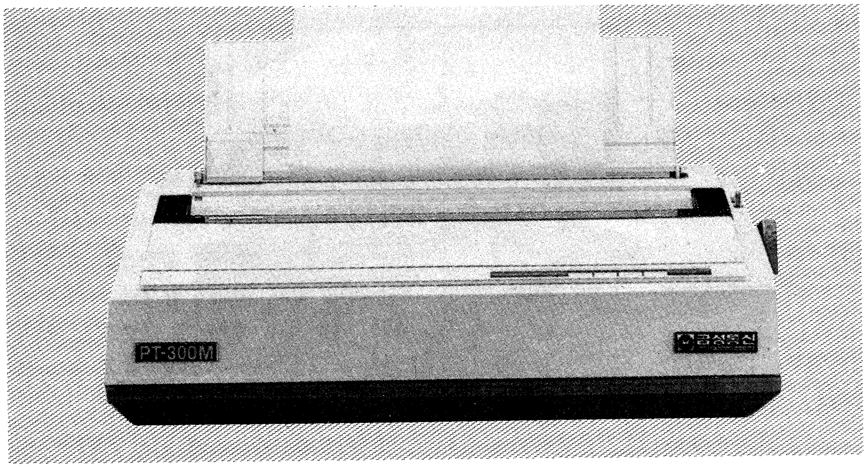
-금성통신

프린터사업 확대

금성통신은 최근들어 퍼스컴의 보급확대와 관련하여 퍼스컴용 프린터의 수요가 크게 늘어감에 따라 이 분야를 주력업종으로 하여 프린터사업을 대폭 강화한다.

금성통신은 현재 전체 매출액의 20% 정도에 불과한 프린터사업을 올해 30%에 이어 오는 90년에는 40%까지 끌어올리는 동시에 생산품목도 다양화하기로 했다. 이를 위하여 금성통신은 1단계로 올해 이 분야에 20억원을 투자하여 생산시설을 확충하고 저가의 신제품 및 제품다양화에 주력할 방침으로 알려졌다.

현재 금성통신은 대용량의 밴드(Band) 프린터로 기존의 B-600, B-1000외에 최근 미국의 DPC사와 기술제휴로 새로운 모델인 초고속, 대용량의 라인(Line) 밴드프린터 LB-600을 선보였으며, 점형태로 글자를 표시하는 24핀 도트매트릭스 방식의 PT-200M, PT-300M외에 자체개발품인 무소음 열전사(Thermal Transfer)방식의 PT-80T 등 다양한 모델을 갖추고 있다.



특히 일본 OKI사와 기술제휴로 최근 개발된 PT-8160은 24핀 도트매트릭스방식으로 국내 저가형 PC를 겨냥한 프린터로서 8K바이트 용량의 자체 데이터저장기능 및 한글/한자를 기본사양으로 하는 등 기능은 보강하면서 가격은 국내 동일기종중 가장 저렴한 경제적인 제품으로 시판할 예정이다.

또한 금성통신은 자체개발와 수출전용 모델로서 계속해서 수요가 증가하고 있는 9핀 도트매트릭스 방식인 PT-160M 프린터를 국내에서도 시판키로 하고, 그동안 한글/한자 등 내수용에 알맞게 기능을 수정, 보완시켜 국내시장에 내놓을 계획이다.

금성통신은 출력장치인 프린터를 중심으로 메모리분야인 FDD, HDD 등도 함께 강화함과 아울러 입력장치인 스캐너까지 생산할 계획이며, 이와 관련해서 지난해 정부로부터 자동자료 처리장비 및 동 부품제조업에 대한 승인을 획득하는 등 컴퓨터 관련산업의 영역을 더욱 확장시켜 주변기기 종합메이커로 성장키 위한 변신을 꾀하고 있다.

-디지탈 테크놀러지

통신카드 및 S/W 개발

디지탈 테크놀러지에서는 MS-DOS에서 IBM XT/AT 및 호환기종 또는 386에서 여러대의 컴퓨터와 통신이 구현되는 통신카드와 S/W를 개발하였다.

이 제품은 스타 LAN방식으로 각 시스템이 워크스테이션으로 각기의 컴퓨테이션(computation)을 할 수 있으며, 초보자도 이용할 수 있도록 운영방식이 메뉴방식으로 되어 있다. 또한 DT.COM이 램상주 프로그램으로 되어 있어 다른 작업중에서도 통신이나 화일전송이 필요할 때 언제든지 통신이 되도록 되어 있으며, RS-232C 통신을 채택하여 접속시스템에서 RS-232C 포트만 있다면 별도의 통신카드를 요구하지 않는다는데 특징이 있다.

초기버전으로 4대, 8대와 통신이 되지만 다음 버전은 16대, 32대, 64대까지 연결하여 통신이 되도록 할 예정이다. 그리고 이때는 RS-422통신을 채택하여 속도와 거리를 넓힐 계획이다.

-(주)럭키

피부구조 측정하는 영상분석기 개발

화장품이 피부에 미치는 영향을 측정하는 영상분석기가 개발돼 피부구조의 변화를 과학적으로 분석할 수 있게 되었다.

럭키가 영국 캠브리지사와 공동으로 제작한 이 영상분석기는 화장품을 사용한 후 피부의 변화과정을 과학적인 데이터로 확인할 수 있는 장치인데 피부상태의 분석, 화장품의 선택, 효과분석까지 가능해 화장품의 신제품개발에도 크게 기여하게 됐다.

이 분석기는 특히 자동회전방식을 적용해 피부샘플을 360°도 회전시켜 전방위에서 피부를 측정함으로써 피부의 주름정도를 입체적으로 측정할 수 있다.

이같은 분석과정에서 화장품은 바르기 전의 피부구조와 바른 후의 피부구조가 데이터로 명확하게 비교돼 화장품사용의 효능, 효과가 정밀하게 나타난다. 이 영상분석기를 이용한 피부구조분석은 종전의 각종 분석기기와 상호보완하여 피지분석, 피부의 표면온도, 탄력, 두께 측정에도 활용함으로써 과학적인 효능을 지닌 화장품의 개발과 함께 피부과학의 발전에도 기여할 수 있게 됐다.

-삼성데이타시스템

미국 SDRC사의 I-DEAS 국내 공급계약 체결

삼성데이터시스템(대표 : 임경춘)은 미국의 SDRC사와 MCAE(Mechanical Computer Aided Engineering)분야의 토탈시스템인 I-DEAS의 국내공급계약을 체결하고 판매, 서비스 및 기술지원에 필요한 교육을 완료하고 활동에 들어갔다.

MCAE분야의 선도적위치에 있는 I-DEAS는 설계, 해석에서부터 실험데이터 분석 시뮬레이션, GNC(Graphic Numerical Control : 그래픽을 이용한 수치제어가공) 데이터 메니지멘트컨트롤, 문서편집 등의 완벽한 기능을 자체에 보유하고 있으며 IBM(RT/PC), HP9000, MICRO VAX, 아폴로 등의 워크스테이션 및 IBM 9370 등의 대형 컴퓨터에서 사용이 가능하다.

또한 삼성데이터시스템은 대형 컴퓨터의 기존 IBM/CAEDS 고객에 의한 기술지원 및 판매도 하고 있다.

-삼성전자
IC카드시스템 개발

삼성전자는 IC(집적회로)를 이용한 카드시스템을 개발하여 공급에 나섰다.

이 시스템은 IC카드에 데이터를 써넣거나 읽을 수 있으며 데이터의 수정 삭제는 물론 IC카드 상호간에 데이터 통신이 가능하며 퍼스컴과도 연결사용할 수 있다.

이 IC카드는 기존의 자기카드가 최대144자까지 기록할 수 있는데 비해 동일한 크기에서도 2천자이상의 정보를 기록할 수 있으며 일단 기록된 정보는 외부에서 쉽게 읽을 수가 없는 것이 특징이다.

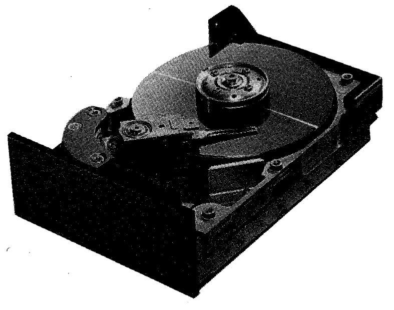
이 제품은 각종 신용카드나 예금통장, 백화점카드, 병원진료카드, 공중전화카드 등의 기능을 1개의 카드에 통합해 사용할 수 있다.

-시게이트 코리아
3.5인치 HDD 국내 공급

시게이트가 내달부터 3.5인치 HDD를 국내에 본격 공급한다.

시게이트코리아(대표 : 경문호)는 올들어 국내 퍼스컴 수출업계가 퍼스컴에 3.5인치 HDD를 채택하여 수출에 나섬에 따라 그동안 거의 공급되지 않던 20, 40MB 제품을 내달부터 월 1만대 이상씩 공급하기로 했다.

시게이트 코리아는 이에 따라 3.5인치 생산라인을 보유하고 있는 시게이트 싱가폴공장에 소요되는 제품의 생산과 공급을 늘려줄 것을 요청했으며 앞으로 폭발적 수요가 예상되고 있는 점을 감안하여 국내공장 설립도 추진중인 것으로 알려졌다.



시게이트코리아가 본격 공급하게될 제품은 ST125(20MB), ST151(40MB) 등 MFM방식으로서 내년 회계년도 부터는 점차 40MB제품의 공급을 늘려갈 계획이다.

-삼성오피스컴퓨터
어음, 수표 발행 시스템 개발

삼성오피스컴퓨터(대표 : 김한기)는 최근 일본ICS사와 기술제휴를 맺고 16비트 퍼스컴과 각종 정보검색시스템을 연결하여 어음 및 수표의 발행에서 처리과정을 완전자동화할 수 있는 어음 및 수표발행 시스템을 개발하였다.

이 시스템은 수취인과 발행인 성명 등 관리정보를 금융기관별로 다양한 모양으로 인쇄할 수 있어 위조를 방지할 수 있고, 각종 정보검색장비를 자유자재로 부착할 수 있어 다량의 자기앞수표 교환접수 및 지급 등의 업무를 신속정확하게 처리할 수 있다. 또한 IBM, FACOM, ACOS 등 대형컴퓨터와 네트워크가 가능해 어음관리를 비롯하여 회계 재무관리 등을 위한 별도의 추가자료없이 수취인 지급은행, 만기일 별로 정보를 출력할 수 있다.

-아폴로컴퓨터
한글시스템 개발

한국 아폴로컴퓨터는 최근 아폴로워크스테이션에 적용되는 한글시스템을 개발하여 한글로 프로그램과 문서작성이 가능하게 되었다.

이번에 개발한 한글시스템은 OS차원에서 한글을 지원하는 한글 DM(Display Manager)에 기초하는 것으로 2종의 한글 비트맵 폰트와 한글벡터폰트 , 한글그래픽리소스(HGPR) 등으로 불리는 응용프로그램 개발툴을 제공한다.

또 워크스테이션 사용자가 직접 한글폰트나 특수문자를 디자인할 수 있게 하는 한글폰트나 특수문자를 디자인 할 수 있게 하는 한글폰트 편집시스템도 제공한다.

정부표준인 KS한글코드 체계(KS5-601) 2바이트 완성형코드를 채택한 이 한글시스템은 컴퓨터 화면을 분할해 사용하는 멀티 윈도우환경에서 한글표시가 가능하며 한글 표준코드로 영문 반각 문자와 전각문자를 동시에 사용할 수 있는 특징을 갖고 있다.

또 별도의 프로그램 없이 한글로 입출력할 수 있으며 그래픽처리 업무도 한글로 수행할 수 있다.

한글폰트편집 시스템을 활용할 경우 점이나 선, 원, 사각형 등을 이용해 문자사이의 간격변화, 문자이미지복사 등도 가능케 된다.

-큐닉스
통합 패키지 SWING V 개발

최근 큐닉스(대표이사 : 이대우)는 일반 상업용 표계산 소프트웨어 SWING CALC와 비지니스 그래픽 소프트웨어인 SWING GRAPH를 개발 완료하여 일반인에 선보일 예정이다.

IBM PC XT/AT 호환용기기 및 자사의 옴니스테이션 시리즈에서 사용할 수 있으며 한글MS-DOS3.2 이상에서 운영되는 이 소프트웨어들은 일상생활에서 발생되는 수치 데이터들의 관리 및 계산을 자동적으로 처리해 주는 기능의 소프트웨어와 이러한 수치데이터들을 그래프 형식의 도표로 표현해 줌으로써 수치간의 비교 및 상황을 더욱 편리하고 효율적으로 분석하는데 도움을 주는 소프트웨어로 이루어져 있다.

이로써 큐닉스는 기존의 데이터베이스 관리시스템 DB 및 관련 소프트웨어인 SCAT(응용 프로그램 자동 작성 시스템)과 CARD(낱장용 데이터 전용 관리 시스템)로 구성된 데이터처리 소프트웨어에 CALC와 GRAPH라는 소

● 미니인터뷰 ●



'딜러 발굴과 그들에게 최대의 기술적 지원이 우리의 역할'

아이엠디 코리아 황윤홍 지사장

호주에 본사를 두고 미국과 아시아 각 지역에 지사를 둔 IMD사(Imagineering Micro Distributors, LTD)의 한국지사인 IMD KOREA가 지난 2월 24일 스위스 그랜드호텔에서 Freelance Plus 3.0에 대한 세미나와 함께 릴리즈 3에 대한 시사와 앞으로 IMD KOREA의 마케팅전략을 발표하였는데 그때 만나 본 황윤홍 지사장은 다음과 같이 밝혔다.

"이제는 한국에도 물건만 공급할 것이 아니라 IMD의 이미지관리나 독자적인 마케팅을 가질 필요가 있다는 IMD 본사측의 결정으로 지난 88년 6월 IMD KOREA가 설립되었습니다. 현재 모두 10명 정도의 작은 규모이지만 우리들의 역할은 직접 엔드유저를 상대하는 것이 아니라 말 그대로 순수 디스트리뷰터입니다. 따라서 국내에서 어떻게 하면 딜러들을 많이 발굴해 낼 것인가와 관계를 맺은 딜러에게는 어떻게든 최대한 기술적인 지원을 해야 하는 것이 앞으로 우리들에게 놓인 과제입니다."

또 "우리 IMD KOREA의 등장으로 인해 기존에 다이렉트로 판매해 오던 딜러측에서 가격적인 면에서 더욱 높아지는 것은 아니냐는 우려로 거부반응을 일으키는 경우가 종종 있는데, 저희는 기존에 딜러들이 직접 본사와 접속함으로써 들였던 시간과 인력을 최대한 줄이면서 동일가나 그 이하로 공급할 것이며 구입후의 기술적인 지원까지도 해줍니다. 따라서 엔드유저에게 공급되는 가격 역시 기존과 별차이가 없고 제품에 대한 신뢰도도 높아질 것입니다. 또한 언제든지 엔드유저나 딜러측에서 요구하는 솔루션을 공급할 수 있읍니다. 단 저희는 엔드유저 상대가 아니라 반드시 딜러를 통해 엔드유저의 요구 또한 듣게 되는 것입니다."라며 IMD KOREA의 역할에 대해 분명한 선을 긋는다.

그리고 IMD KOREA의 소프트웨어 불법복제에 관한 앞으로의 조치 역시 주관심이 되고 있는데 이에 대해 황윤홍지사장은 "본사에서는 강경한 자세를 취하길 원한고 있읍니다만 강경한 자세를 취해 금방 해결된 문제는 아닌 것 같읍니다. 다만 우리들로서는 유저들이 조금더 가격을 주고 구입함으로써 세미나나 자문 트레이닝 등 테크니컬 트레이닝 기회를 되도록 많이 부여 받게 해주고 계속해서 업데이트되는 제품을 커버할 수 있도록 방법을 제시해 주고자 합니다. 따라서 이 방법이 장기적으로 휠씬 경제적이라는 인식을 유저가 가질 수 있도록 유도하는 길밖에 없는 것 같습니다. 그러나 역시 중요한 것은 유저 자신이 이제는 소프트웨어 개발을 인정해 주고, 이를 합당한 가격을 지불하고 구입해야한다는 인식을 하루빨리 가져 주어야 이 문제는 해결되리라 봅니다."라고 말한다.

앞으로 발표될 LOTUS 1-2-3의 새로운 버전인 릴리즈 3의 경우 릴리즈 2(lotus 1-2-3)에 비해 어떤 점들이 개선되었는지에 대해 "릴리즈2에 비해 더 전문적인 기능을 원하는 유저를 타켓으로 한 제품으로, 기존 릴리즈2가 8192×256셀의 한 워크시트를 핸들링한데 비해 Z축까지 3차원 지원이 가능해 8192×256셀의 워크시트를 256개 동시에 핸들링할 수 있읍니다. CPU는 80286이상이어야 핸들링이 가능하고, DOS와 OS/2 모두 선택이 가능하고, 속도면에서도 2차원에서 256개의 워크시트를 수작업하는 것보다는 빠릅니다."라고 한다.

또한 12개의 형식(Pie, Bar, Stacked Bar, Line, Bar Line, XY Scatter, Table, High-Low-Open-Colse, Area Formats)이 이미 만들어져 있어 데이터만 입력시키면 자동적으로 연결되어 그래프를 제작할 수도 있다고 한다.

국내에는 올 6월쯤 공급될 예정인데 원래 보도에 의하면 1년전에 공급될 예정이었으나 완벽한 제품을 내놓겠다는 LOTUS 본사의 속사정(?) 때문에 지연되었다고, 현재도 선정된 유저를 통해 베타테스트중이다.

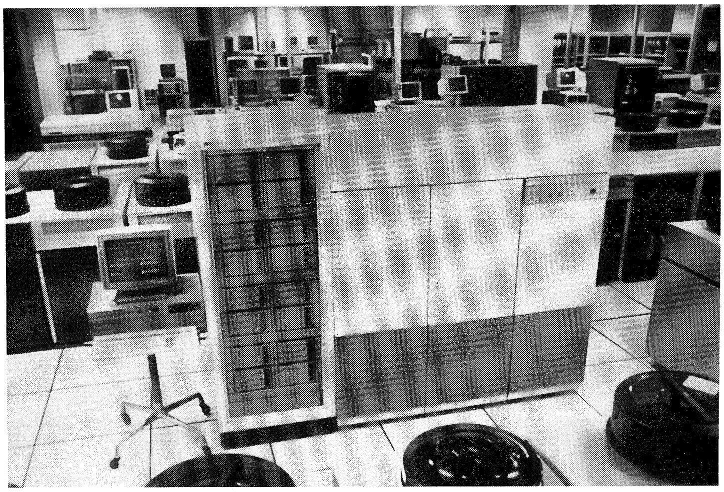
그러나 6월에 보급되어도 한글지원이 불가능해 아쉬움을 남기고 있는데 황윤홍지사장은 "이번 세미나를 준비하는 과정에서 LOTUS본사와 IMD 본사에 강력하게 요구하였고 그들도 이의 필요성을 인식하고 있습니다. 현재는 한글지원 개발방법을 어떻게 추진할 것인지가 검토되고 있습니다. 그 검토가 끝나는 즉시 채택된 방법에 의해 실행하여 올해안으로 결정되지 않을까 생각합니다"라고 밝힌다.

앞으로 계획에 대해 "우리는 딜러와 공존해야 하기 때문에 우선은 국내의 딜러들과 아이템별로 원활한 관계를 맺어 마케팅에 대한 적극적인 지원을 할 생각입니다. 또한 아이템에 관계없이 정보채널의 네트워크나 이미지광고를 확대할 예정인데 현재 LOTUS의 경우는 유저그룹이 형성되고 있는데 이들 그룹이 좀더 활성화될 수 있도록 적극적인 지원을 할 것입니다."

또한 밀려드는 국제적인 압력에 의해 우리나라가 반드시 어느 정도 한정된 양을 수입해야 한다면 소비성 있는 것보다는 기술발전에 도움이 될 수 있는 분야를 대신한다면 국가의 장래를 볼때 빙산의 일각을 양지로 끌어 낼 수 있는 것이 아니겠냐며 말을 맺는다.

프트웨어 두가지를 추가시켜 강력한 데이터처리 소프트웨어의 통합 패키지를 갖추게 되었으며, 일반 사용자들은 데이터 처리를 위한 다양한 기능들을 한번에 제공받을 수 있게 되었다.

금년 상반기부터 이들 5가지의 소프트웨어(SWING DB, SWING SCAT, SWING CARD, SWING CALC, SWING GRAPH)를 하나로 패키지화하여 SWING V라는 통합 소프트웨어로 일반 사용자에게 제공할 계획이며, 기존 SWING사용자를 위하여 SWING PARTNER(SWING CALC, GRAPH) 로도 제공할 예정이다. 따라서 일반 사용자들은 데이터베이스 관리 및 표계산, 데이터의 그래픽화 등 다양한 소프트웨어를 통하여 더욱 광범위하고 효율적인 데이터 처리를 할 수 있게 되었다.



-왕컴퓨터
수퍼 미니컴퓨터 VS10000시리즈 발표

왕컴퓨터 코리아는 고속 연산처리에 적합한 VS10000 수퍼미니컴퓨터 시리즈를 발표했다.

모델 50, 75, 100으로 구성된 VS10000 시리즈는 첨단 ECL 기술의 이용과 새로운 디스크 서브시스템, 시스템관리 소프트웨어의 채택으로 다량의 상업용 데이터처리는 물론 통합화상 정보처리와 같은 사무자동화 어플리케이션 사용시 성능이 뛰어나다. 또한 10진연산과 부동 소숫점, 수 연산기구를 모두 지원하는 산술연산 장치가 있어 COBOL, COBOL 85 등으로 작성된 업무용으로 사용시 처리속도가 빠르다. 새로 개발된 디스크 서브시스템은 약 0.56㎡에 170억 바이트의 정보를 저장할 수 있어 기존의 시스템보다 10배 이상의 고밀도로 정보를 보관할 수 있다.

VS10000 시리즈는 IBM370 명령어에 기반을 두고 있는 VS명령어 세트를 사용하여 ECL 기술, 하드웨어 가속기의 채택, 명령의 파이프라이닝, 마이크로코드의 최적화 등이 가능하게 하였고 이러한 기술의 지원으로 생산비용이 절감되고 시스템의 안정성이 높아졌다.

VS10000은 모든 VS시스템과 호환성을 갖고 있어 VS5000 및 VS7000 시리즈에서 사용하는 모든 소프트웨어 및 주변기기들을 수정없이 그대로 사용할 수 있다.

-FKL동화
10만원대 보급형 프린터 개발 예정

FKL동화는 최근 컴퓨터 프린터시장에 참여하는 한편 프린터용 헤드의 국내시판에도 적극 나서기로 했다.

각종 헤드생산업체인 FKL 동화는 최근 9핀 도트매트릭스 프린터의 개발을 끝내고 지난해 새로 완공한 건물에 곧 조립라인을 설치하여 생산을 하기로 했다. 현재 개발한 프린터는 숫자 및 영문표기가 가능한 것인데 FKL동화는 이 제품을 대기업을 통해 OEM수출하는 한편 일본시장에 직접 수출도 모색하고 있다. 또한 한글표기가 가능한 9핀 프린터를 개발하여 국내에 시판하기 위하여 큐닉스와 공동으로 한글화작업을 진행하고 있으며 이 작업이 완료되면 10만원대의 프린터를 싼 가격에 공급할 예정이라고 회사측은 밝혔다.

FKL동화는 이 프린터를 개발하기 위하여 일본에서 설계기술을 도입하였으며 부품개발은 자체기술로 해결했다.

이 회사가 계획하고 있는 프린터 생산능력은 월 4만대 수준이다.

한편 FKL동화는 그동안 전량 일본에 직수출해 왔던 프린터용 도트매트릭스 헤드도 국내시판 하기로 했다.

-큐닉스
AutoCAD용 프린터 드라이버 개발

큐닉스(대표 : 이대우)는 Auto CAD의 출력을 자사의 프린터로 할 수 있는 프린터 드라이버를 개발하였다.

현재 Auto CAD내에서는 레이저빔 프린터를 지원하기 위하여 휴렛팩커드사의 레이저제트 플러스용 드라이버를 이용하고 있으나, 이것은 이미지로 처리하기 때문에 출력하는데 상당한 시간이 소요되고 있다.

이러한 단점을 보완하기 위하여 큐닉스는 Auto CAD의 제작회사인 Autodesk사 의 ADI규격을 이용하여 레이저빔 프린터를 플로터처럼 사용할 수 있게 함으로써 출력속도를 향상시

켰으며 또한 시간과 비용을 절감할 수 있게 되었다. 그리고 자사의 도트 매트릭스 프린터도 완벽하게 지원할 수 있는 드라이버도 개발 완료한 것으로 알려졌다. 또한 현재는 MS-DOS환경하에서만 사용할 수 있으나 앞으로는 다른 환경도 계속 지원할 계획으로 알려졌다.

-한국IBM
한글 표준코드 변환 프로그램 개발

한국IBM은 IBM370 계열의 중·대형시스템 및 S/36, S/38, AS/400 등 중·소형시스템을 위한 한글표준코드 변환 프로그램을 발표했다.

한국IBM은 지난 1988년 11월 IBM 5550/5540 계열제품에서 한글표준코드를 지원하기 위해 한글표준코드의 모든 지원문자와 코드변환 프로그램을 발표했으며, 이 프로그램은 이미 출하된 바 있다.

지난 87년 5월 정보교환용 한글 표준코드가 공업진흥청에 의해 제정 발표됨에 따라 한국IBM이 정부가 제정한 한글표준 코드를 적극 지지하고 이를 IBM 시스템에 채택하기 위한 일환으로 개발한 한글표준코드 변환 프로그램은 앞으로 IBM기종의 사용자들에게 많은 도움이 될 것이며 한국IBM의 한글표준코드 지원의지를 확고하게 보여주는 것이다. 그리고 한국IBM은 기존의 IBM프로그램을 사용하고 있는 고객에게 추가부감없이 해당코드변환 프로그램을 공급하기로 했다.

-효성중공업
FA사업 집중 육성방침

중전기 전문업체인 효성중공업이 고성장, 고부가가치 제품생산을 위해 FA사업을 집중육성하고 있다.

효성중공업(대표:김인환)은 지난해 7월 초고압분야의 산업합리화조치 전면해체로 지금까지 한국전력에 독점공급해 왔던 중전기 사업전망이 불투명해 짐에 따라 기업변신차원에서 FA사업을 적극 추진하고 있다는 것이다.

이를 위해 효성중공업은 기존의 전기제어사업부에서 한정적으로 벌여왔던 FA사업을 전기제어사업부를 주축으로 각 사업부가 전문분야 별로 신규사업을 벌이는 한편 기술연구소가 이를 강력히 지원하는 등 전 사업부가 FA사업육성에 적극나서고 있다.

효성중공업은 계장설비 분야에서는 FMS(복합생산시스템) 구축을 가능케 하는 프로세스용 LAN"ADMAP"를 개발하여 플랜트관리로부터 조업, 제어에 이르기까지 토탈 제어시스템을 90년 발표할 계획이다. 또 기존 생산해 왔던 단일 루프제어기 및 온도제어기 등 단품계측기의 생산을 더욱 확대할 방침이다.

PLC분야에서는 국산화에 성공한 소형 HX시리즈의 양산과 함께 중형PLCEX 시리즈의 국산화도 추진키로 했으며 소프트웨어개발을 위한 전문팀을 구성 컴퓨터뱅크를 이용한 시스템개발 등 PLC를 응용한 어플리케이션사업을 대폭 강화할 계획이다.

-한국IBM
신모델 ES/3090 및 ES/4381 발표

한국IBM(대표:피터 로울리)은 대형시스템이 팽창하는 업무량을 효율적으로 관리할 수 있도록 대형시스템의 정보관리 및 처리능력을 향상시켰다고 발표했다.

한국IBM에 따르면 소프트웨어의 향상, 새로운 대형 기종의 추가발표 및 기억장치의 가격인하 등을 통해 업계 최첨단의 대형시스템 아키텍처인 ESA/370이 고객에게 더욱 높은 생산성과 저렴한 가격으로 제공되게 되었다.

새로운 주전산기 모델 ES/3090모델 100S와 ES/4381모델그룹 90E는 동기종의 모델 가운데 최하위 모델로써 저렴한 가격으로 업계 최첨단 설계인 ESA/370과 그 운영체계 MVS/ESA의 장점을 최대로 이용할 수 있는 기회를 제공하게 된다.

-한국전산업협동조합
수주확대에 공동 대처하기로

키펀치용역 업체들은 올해 해외시장 개척 및 중소업체간 협력체제 구축 등을 통해 수주확대에 적극나서기로 했다.

한국전산업 협동조합(이사장:박봉규)은 최근 정부 및 투자기관의 전산화계획이 활발히 추진되면서 키펀치 및 소프트웨어개발 용역이 늘어나자 올해 전산용역 수주실적 목표를 33억원으로 책정하고 다각적인 수요확대 방안을 강구하고 있다는 것이다.

전산산업 협동조합은 특히 엔화강세, 인력부족 등으로 일본 대형컴퓨터 업체들에 대한 협력요청이 날로 늘고 있는데다 유럽 및 동남아시아 컴퓨터 업체들의 관심도가 높아지고 있는 점을 감안하여 외국업체와 기술제휴를 맺는 등 상호협력체제를 구축하여 해외 수출시장을 확대해나가기로 했다.

이와함께 지방자치제 실시와 더불어 폭주할 지방자치기관의 전산수주 용역을 겨냥하여 부산, 대구, 광주 등 지방 대도시 소재 키펀치용역업체들의 조합 가입을 적극 유도하는 한편 중소업체간의 수평, 수직적 협력체제를 구축하여 공동개발을 통해 채산성을 한층 높여갈 계획이다.

한국전산업 협동조합은 현행 사무국위주의 조합운영체제를 소프트웨어 및 키펀치 용역 분과위원회 중심으로 전환하고 중소기업 지원자금, 공제사업기금을 효율적으로 운영하고 중소업체들의 경영활성화를 촉진할 방침이다. PC

소액 리스정책, 국내기종 보호위해 개선되어야 한다

88년도 정보기기 품목별 내수 현황에 따르면 국내 정보산업기기 관련 총생산액 1조7천9백22억원중 3천54억원인 17.0%가 국내에 판매되어 전년대비 64.7%가 증가하였고, 이중 16비트 PC의 판매가 전년대비 66.9%, FDD는 461.1%, 모니터(모노)는 408.7%로 고성장을 보이고 있는 가운데 리스를 통한 컴퓨터도입이 크게 늘고 있어 주목을 끌고 있다.

특히 한국전자공업진흥회가 국내 14개 리스사를 대상으로 분석한 '컴퓨터류 리스지원 실적'에 따르면 국산컴퓨터보다 외국산컴퓨터도입에 더 많은 리스지원이 이루어져 문제시되고 있다.

지난해 리스지원액은 모두 2천5백98억9백만원으로 전년도 지원인 1천9백79억7천1백만원보다 31.2% 증가하였으나 이중 국산컴퓨터의 도입에 지원된 금액은 1천2백32억5천5백만원으로 전년대비 16% 증가에 그쳤다.

반면에 외국산컴퓨터의 도입에 지원된 금액은 1천3백65억5천4백만원으로 지난 87년에 비해 48.9%나 신장된 것으로 집계되었다.

또한 지난해 국산컴퓨터류 리스자금중 국민투자기금의 사용액은 2백13억6천6백만원으로 전체 지원금의 17.3%를 차지한 것으로 나타났다. 이는 지난 87년의 1백43억3천8백만원보다 49%가 증가된 것이나 총판매액 3천54억원중 국민투자기금으로 구입한 금액은 불과 전체의 7.0%인 2백14억원으로 지난 87년의 7.7%에 비해 0.7%가 감소한 것이고 사용금액도 시장 증가율인 64.7%에 못미치는 수치이다.

2백14억원의 국민투자기금 사용금액중 PC(Personal Computer) 는 지난해보다 106.5% 증가한 45억1천4백만원으로 전체의 21%를 차지하여 가장 활발히 사용되었고, CRT터미널은 168.2%로 35억1천4백만원이고, 프린터는 114.8%로 42억7천1백만원, 모뎀은 90.6%로 8억3천3백만원, 마이크로급은 183.3%로 7억7천9백만원이다.

반면 미니급은 −19.4% 감소한 25억7천7백만원을 사용하였고 소프트웨어의 경우는 사용률은 48.8%가 증가하였으나 사용금액이 3억1천1백만원으로 저조한 실적을 보였으며 나머지 기타 H/W품목은 −4.4% 감소로 45억3천7백만원이 사용되었다.

이와 같이 활발히 진행되고 있는 리스를 통한 컴퓨터관련기기 도입이 국산기종보다 외국산기종에 치우치는 이유는 14개의 리스사마다 배당된 지원금액으로 보통 소형이하인 국산컴퓨터의 소액리스를 여러번 처리하는 것보다는 거액을 필요로 하는 중형이상인 외국산컴퓨터를 도입하는 것이 번거롭지 않다는 논리에서 이다.

왜냐하면 보통 소액리스의 경우 리스이용자들이나 리스사가 거래은행에 제출할 서류가 많고 인출절차가 복잡하기 때문에 소액리스인 경우에는 빈번하게 그 복잡한 절차를 거치는 것보다는 아예 직접거래를 하는 것이 낫다는 견해에서 이를 기피하기 때문인 것으로 풀이된다.

따라서 이의 해결책으로 한국전자공업진흥회측은 국산컴퓨터류로 배정금액을 약 8백억원선으로 정해 타기기류로의 인출을 금지하여 각 리스사별로 국산컴퓨터의 리스영업을 유도할 것과 소액리스에 대한 인출절차를 엔드유저가 리스사에게 제출하는 서류의 경우에는 현재의 25종목에서 7종목으로, 공급자가 리스사에 제출하는 서류의 경우는 현 15종목에서 8종목으로, 리스사가 거래은행에 자금인출 요청시 제출하는 서류는 현 12종목에서 6종목으로 축소할 것을 건의하고 있으나 아직 확정되지는 않고 있는 실정이다.

그러나 지금과 같은 추세로 16비트 PC구입자가 늘어난다면, 약3년후에는 정보산업이 어느 정도 발전된 단계에 돌입할 것이고, 그렇게 되면 이 소액리스에 관한 문제는 자연적으로 해결되지 않을까 한다. 결국 정보산업이 어느 정도 활성화 되기까지는 14개 리스사의 역할이 그 어느 때보다도 중요하다는 말이다. 따라서 국내의 14개 리스사에서는 국내 정보산업을 보호하는 의미에서 적정분배선까지는 힘들더라도 소액리스를 찬밥(?) 취급하지 말아야 할 것이며 또 정부차원에서도 한국전자공업진흥회의 의견을 잘 수렴하여 인출절차를 좀더 간소화하고, 당분간이나마 국산컴퓨터 리스에 대한 지원을 강화해야 하지 않을 까 한다.

국내 정보산업을 보호육성하는 길은 어느 개인이 노력한다고 될 일이 아니라 우리 모두가 힘써야 할 일이다.

〈글·김현정기자〉

행정전산망, 시작부터 문제점 많다

정부는 지난해부터 금융, 교육, 국방 등을 국가기관전산망으로 통합하여 추진하고 있다. 이와 함께 중추적인 행정전산망 구축에 91년까지 2천31억원을 투입할 예정이다.

정부의 이러한 계획은 주민관리 및 부동산, 자동차, 통관업무, 고용관리 등을 주무부처별로 전산화하여 이들 정보를 전국 방방곡곡 어느 곳에서나 쉽게 이용하는 신속함과 정확함을 꾀하는데 있다.

그런데, 행정전산망 구축을 위한 워크스테이션이 정보의 저가구매정책으로 수차례의 입찰과 유찰을 거듭했었다.

그러던 정부의 구매입찰이 수차례의 팽팽한 줄다리기 끝에 지난 2월 27일로 모든 시·도의 워크스테이션 납품업체선정을 자체조정을 통하여 마쳤다.

각 시·도가 선정한 업체를 알아보면 다음과 같다.

먼저, 지난해말에 금성사(광주시), 대우통신(경남), 일진전자(전북)가 계약을 체결했고, 지난 2월 25일에는 9개 시·도별 공급선정업체로 현대전자(서울시), 삼보컴퓨터(경기도), 테레비디오컴퓨터(대구시), 동양시스템산업(인천시), 일진전자(충남 및 대전시), 대우전자(전남), 삼성전자(경북), 금성사(강원도)가 27일에는 동양나일론(충북), 큐닉스(부산시), 고려시스템(제주도)이 계약을 체결하였다.

이들 업체들은 3월 4일까지 조달청과 정식계약을 체결하고 60일내에 전국 15개 시·도의 읍·면·동사무소에 워크스테이션을 설치하면, 8월말부터는 전국적으로 행정전산망이 구축될 것이다.

수의계약을 통하여 구두로 계약이 체결된 공급가격은 정확히 밝혀지지 않으나 대부분 정부가 제시한 구매예산보다 저렴한 80% 내외로 알려졌다.

이에 대해 각 업체들은 조달청 구매예상가의 90% 수준까지 높이기로 공동대처했었다. 그러나 일진전자가 이를 파기하고 계약을 한후에 잇달아 현대, 삼성, 삼보 등이 계약을 체결했다.

이러한 정부의 저가구매정책으로 인해 국가적인 사업이 어쩔 수 없이 업체들에게 무리를 안겨주어 좋은 성능과 기능을 갖춘 기기보다 가격에 맞추다보니 기대했던 제품보다 성능이 떨어진 제품을 납품하려는 움직임을 보이고 있어 앞으로의 귀추가 주목되고 있다.

구체적인 한 일예로서 정형파 UPS 대신에 구형파 UPS를 사용하려는 움직임을 들 수 있다.

구형파 UPS는 대개 전압이 110V 전용으로 설계되어 있어 220V를 사용하는 지역에서는 별도의 변압기를 설치해야 하는 번거러움 뿐만 아니라 순간정전시 과도전압 및 노이즈전류가 발생하여 데이터 손실과 파괴를 가져오는 위험성을 안고 있다.

반면에 정현파 UPS는 정전시에도 정형파를 출력하고, 일정한 전압을 유지하여 컴퓨터가 데이터손실을 할 위험성이 구형파 UPS에 비해 현저하게 적다.

이런 성능차이가 있는데 앞으로 각 제품에 어떤 것이 채택되어 운용될지 자못 궁금하기 짝이 없다.

또한, 행정전산망의 주컴퓨터인 미국의 톨러런스사의 이터너티기종이 시범운행(국민연금관리공단의 주컴퓨터로 사용)과정에서 잦은 고장을 일으켜 다른 회사의 제품으로 교체될 예정이며, 앞으로 행정전산망에서도 문제시 되고 있는 형편이다.

정부는 워크스테이션공급 주관기관인 '한국컴퓨터연구조합'측과 협의하여 OS는 미국의 SCO사의 SCO-XENIX를 사용하고, 데이터베이스는 인포믹스사의 C-ISAM을, 데이터백업을 위한 보조기억장치(STD)는 어윈사의 제품을 선정했다.

또한 행정전산망 워드프로세서 구매방식을 다음과 같이 바꾸기로 결정하였다. 즉, 예전에는 금성 소프트웨어와 독점구매방식을 취했으나, 이를 수정하여 하나워드프로세서와 100% 데이터 호환이 되는 워드프로세서를 행정전산망용으로 추가선정하기로 하였다. 이는 워드프로세서의 품질향상과 제품의 다양화를 위해 긍정적인 반응을 얻고 있다.

이런 변경사유의 이면에는 하나워드프로세서만의 사용으로 인한 행정공무원의 획일화된 워드프로세서 사용을 지향하고, 자유경쟁체제를 유도하여 국내 워드프로세서의 기능을 고도화하는데 의미가 있다. 그러나, 이제 시작단계에 있는 행정전산망에 여러 워드프로세서가 사용되고, 이로인해 사용에 있어 혼란이 가중되는지의 여부도 유념해야 할 것이다.

어쨌든 이제 수차례의 난항을 거듭한 행정전산망 구축계획이 정식계약을 체결했고, 앞으로는 이의 설치 및 운용의 문제가 남았지만 계획된 예정대로 착오없이 잘 마무리되어 국민들의 생활에 편리함을 제공했으면 한다.

〈글·김용기자〉

서클동정

●계명대 컴퓨터 그래픽반●
서울 전시회로 새로운 도약 다짐

컴퓨터 그래픽의 심미적인 기능을 강화하여 새로운 장르의 예술로 승화시키겠다는 의지로 계명대학교 미술대학 컴퓨터 그래픽반이 창설된 것은 88년 8월이었다.

동 써클은 현재 배인호교수의 지도 아래 10명의 회원으로 구성되어 있으며, 매주 1회의 정기적인 모임을 통하여 작품과 컴퓨터 그래픽에 관한 지식을 토의하고 있다. 시스템이 1대밖에 없어 사용하기에 어려움을 겪지만 지난 88년 12월에는 대학교 써클로서는 처음으로 대구에서 제1회 전시회를 개최하고, 지난 2월 18일부터 22일까지 서울에서 2회 전시회를 개최하는 등 활발하게 활동하고 있다. 회원은 계명대학교 응용미술학과생으로 제한하고 있다.

노성진회장(응용미술학과 4)은 서울 전시회에서 많은 호응을 얻어 보람을 느끼며 이를 계기로 회원들이 작품활동에 전념해 여름전시회는 색다른 변모를 보이도록 노력할 것이라 한다. 또한 현재까지는 2차원 그래픽을 주로 다루었지만 곧 3차원 그래픽이 가능한 소프트웨어를 구입하여 응용할 수 있도록 할 계획이라고 한다.

동 써클에 관한 문의사항은 전화번호 (053) 620-2383 계명교 대학미술대학 컴퓨터 그래픽반으로 하면 된다.

서울 전시회에서 얻은 호응으로 더욱 예술적인 작품을 만들겠다는 의지를 다지게 되었다고.



●대한치과의사 컴퓨터 동호인회●
교육 프로그램 개설

대한치과의사 컴퓨터 동호인회에서는 지난 2월 8일에서 시작하여 5월 15일까지 치과단위로 컴퓨터를 사용하는 방법 일반에 관한 3개월간의 교육프로그램 과정을 개설하여 교육하고 있다.

이번에 개설한 제1기 교육프로그램의 과정은 컴퓨터의 활용과 현황, 프린터 사용법, MS-DOS, 보석글 사용법, PFPW, DB Ⅲ+, 로터스, 공중정보통신망, 치과전용의 패키지의 소개와 질의 응답시간으로 구성되어 있다.

자세한 문의사항은 전화번호 543-0219 박은기 치과로 하면 된다.

●동석PC클럽●
사내 PC광들의 모임

동석 PC클럽은 사내의 PC 동호인들간에 정보의 교환과 PC조작능력의 배양, 자체 프로그램 개발, 친목도모를 위해 지난 88년 12월 28일에 창립총회를 가졌다.

동 클럽은 월2회의 교육을 통해 MS-DOS, PCTOOL, 워드프로세서, 로터스 등에 관하여 토론 및 정보교환을 하고 있으며, 현재 회원은 17명으로 회장과 총무가 동 클럽의 운영을 담당하

고 있다. 조현만회장(생산부 차장)은 앞으로 클럽 전용의 PC를 구입하는 것과 전문강사를 초청, 강연을 실시하여 컴퓨터에 관한 지식을 넓혀가며 자체 프로그램개발에 중점을 두고 노력할 계획이라고 한다.

동 클럽에 관한 문의사항은 전화번호 (0562) 72-0451~9 포항 동양석판공업(주) 설비부 김재영씨 앞으로 하면 된다.

● DTUC ●

실습실과 도서실 마련계획

컴퓨터를 배우는데 있어 사용자들이 효과적으로 사용할 수 있도록 대학생 4명이 모여 지난 2월 1일 Digtal Talks User Club이란 유저클럽을 창설하였다.

현재 회원이 10명에 이르고 있는 동 클럽은 대표(윤일호 : 단국대 물리학과 3), 기획, 총무, ID실장 등 4명의 운영위원으로 구성되어 있고 아직은 정기모임을 가지지 않았지만 어느 정도 회원이 확보된다면 분기별로 모임을 가질 것이라고 한다.

DTUC는 현재의 사무실이 위치한 곳에 IBM 호환기종 컴퓨터를 16대 설치하여 회원들이 컴퓨터를 필요할 때마다 사용할 수 있는 실습실을 마련하는 것과 컴퓨터자료를 배치한 도서실을 마련하여 회원 누구나 필요한 자료를 열람할 수 있도록 하고 스터디 모임을 활성화하는 것과 BBS를 개설하여 신속하게 정보를 교환하려는 계획을 가지고 있다.

동 클럽에 관한 문의사항은 전화번호 (02) 479-2001/2211 DTUC 앞으로 하면 된다.

● CACC ●

제2회 연합전시회 마쳐

전남지역 컴퓨터 써클연합회는 지난 3월 10일부터 12일까지 광주 학생회관 로비에서 제2회 연합 컴퓨터전시회를 주최하였다. 동 전시회는 하드웨어 3작품, 소프트웨어 17작품 등 20작품이 선보여 많은 호응을 얻었다.

동 연합회는 지난 85년 10월에 전남지역 대학교 컴퓨터써클간의 신속한 정보교환과 학술적인 발전, 유대강화를 위해 창단되었다.

현재 광주경상대, 광주교육대, 목포대, 송원전문대, 전남대, 조선대, 조선대병설 공업전문대, 호남대 등 8개 학교의 1,000명의 회원으로 구성되어 있다.

활동사항으로는 매년 연합 컴퓨터 전시회를 개최하고 컴퓨터 지식의 전달과 새로운 정보의 교류를 위해 분기별로 학술 세미나를 실시하고 있다. 또한 회원교간의 유대강화를 위해 체육대회도 개최하고 있다.

이진윤 회장(전남대 전기과 3)은 타지역의 컴퓨터써클 연합회와 좀더 활발한 교류를 위해 노력할 것이며 무엇보다도 회원들의 교육에 역점을 두고 학술행사를 강화할 것이라고 한다.

동 연합회에 가입하려면 동 연합회와 같은 목적을 지닌 전문대 이상의 학교 컴퓨터 써클(한 학교에 하나의 써클)로 연합회 회장단의 만장일치의 찬성으로 가입할 수 있고, 회원이 되면 학교당 월 5천원의 회비를 납부하면 된다. 동 연합회에 대한 문의사항은 전화번호 (062) 92-6834 이진윤회장 앞으로 하면 된다.

● MEDICOM ●

신입생 교육과 전시회 준비에 박차

MEDICOM은 조선대학교 의대, 약대, 간호대의 학생들이 모여 지난 83년에 컴퓨터의 효과적인 활용을 위해 형성한 써클이다.

현재 의대 44명, 간호대 15명, 약대 8명의 회원이 가입하고 있으며, 회장·부회장·총무와 편집·관리·교육·기획·개발부장이 임원진으로 구성되어 있다. 회원의 교육은 신입생과 기존의 회원으로 나누어 MS-DOS와 파스칼, 유틸리티에 관하여 실시하고 있고 학기중에는 매주 토요일 모임을 가지고 있다.

김경모회장(의대 본과 2)은 신입생 교육을 강화하고 4월말에 있을 전시회 준비를 더욱 철저히 하여 실력을 갖춘 MEDICOM이 되도록 노력할 것이라고 한다.

동 써클에 관하여 알고 싶은 분은 전화번호 (062) 56-3375 김경모회장 앞으로 하면 된다.

● On-line BBS ●

컴퓨터로 대화의 창 마련

PC 유저들간의 PC를 통한 대화를 목적으로 한 개인 BBS가 지난 3월 11일 한 개인(손희권 : 성균관대 전자과 4)에 의해 개설되었다.

동 BBS는 개방 시스템으로 사용자는 컴퓨터와 모뎀, 전화만 있으면 누구든지 사용할 수 있다. 운영 프로그램은 'Wildcat'이며 IBM PC / AT에 40MB의 HDD를 부착하고 있다. 운영 시간은 매주 월요일부터 토요일까지 오후 9시 30분부터 12시까지이며 하루에 개인이 이용할 수 있는 시간은 20분으로 제한되어 있다.

동 BBS를 이용하게 되면 공개 소프트웨어를 공급받을 수 있으며, 유저가 게시판을 사용할 수 있어 컴퓨터에 관한 여러가지 문제점을 게시하여 해결할 수 있다.

손희권씨는 현재 실험단계인 RBBS를 분석해 한글 BBS로 실현하여 사용자가 더욱 편리하게 이용할 수 있도록 할 계획이라고 한다. 동 BBS에 관한 문의사항은 전화번호 (02) 603-1835 손희권씨 앞으로 하면 된다.

국외소식

텍사스 인스트루먼트사

16M비트 DRAM 히다찌와 공동개발

히나찌제작소와 미국 텍사스 인스트루먼트사는 다음세대 반도체인 16M비트 DRAM을 공동개발하는 계약을 체결하였다고 발표하였다. 양사 공동으로 개발에 몰두함으로써 기술개발을 스피드업시켜 타사에 앞서서 제품화를 꾀하기 위해 체결한 이번 계약내용의 골자는 양사는 16M비트 DRAM의 개발을 진행하기 위해서 ① 현재 각사가 보유한 개발 및 제조기술을 서로 공유한다. ② 정보기억부분에 상당하는 셀구조 등의 기술을 공동개발한다. ③ 제품화는 각각 따로따로 한다-등. 공동개발 단계에서 새삼스럽게 생기는 특허기술 등에 대해서는 공유로 하는 것으로 합의하였다. 계약기간은 3년이며 앞으로 16M비트 DRAM만이 아니고 그 다음 세대제품에 대해서도 공동개발의 길을 찾아 갈 방침이다.

마이크로시스템즈사

무게가 2.5kg에 불과한 랩탑기 발표

마이크로시스템즈는 지난 1988년 11월에 무게가 2.5kg에 불과한 랩탑기 'The Book'을 개발하였다. 본체크기는 A4판사이즈이고 두께는 47㎜이며, IBM PC/XT와 호환성을 지니고 있는 동기종은 CPU로 V30(8/4.77MHz)을 채용하였으며 주기억용량은 640K바이트이다(표 1 참조). 플로피디스크 대신 1M바이트의 메모리카드를 사용한 동기종의 가격은 우리나라 돈으로 약 130만원 정도이다.

디스플레이는 반사형 STN액정판넬을 사용하였으며 사이즈는 약 9.6인치, 화면표시는 640×200 도트이다. 메인보드는 2장의 8층기판으로 구성되어 있어 점유면적이 본체의 1/2 밖에 되지 않는 동 제품을 마이크로시스템즈에서는 올해안에 약 절반으로 줄인 새로운 모델을 발표할 예정이다.

보조기억장치로는 메모리 카드(최대 1M바이트)와 IC카드(2K 또는 8K바이트)를 사용하며 2400비트/초, MNP 클래스4의 모뎀을 본체내에 표준탑재하고 있다. 또한 키보드 왼쪽안에 박스형의 확장유니트를 장착할 수도 있게 되어 있으며 표준으로 백업전지가 내장되어 있다. 앞으로 3270단말 에뮬레이터나 팩시밀리 어댑터 등의 유니트도 발표할 예정이다.

〔표 1〕 The Book의 규격

CPU 수치연산 프로세서	V30(8M/4.77MHz) 장착불가능
주기억 ROM RAM디스크	640k바이트, BIOS용 64k바이트, MS-DOS 등 512k바이트 64k바이트
디스플레이장치 그래픽스표시	STN방식액정 640×400도트
내장 인터페이스	RS-232C×1, 디지탈 RGB(CGA)×1, 센트로닉스규격×1, SCSI×1, FDD인터페이스(2DD/2HD)×1, 전화회선 모듈러 체크×2
슬롯	메모리 카드용×2, IC카드용×1, 확장 유니트용×1
OS	MS-DOS3.1(ROM)
배터리 운용시간	60분(내장 NiCd), 180분(전지유니트 장착시)
외형치수/중량 출하시기	297×210×47㎜(WDH)/2.5kg 1988년 11월

후지쯔

텍스트를 음성으로 변환하는 소프트웨어 개발

후지쯔는 시각장해자지향의 음성합성 소프트웨어 「FMTALK팩」을 발표하였다. 이것은 텍스트를 음성데이터로 변환하여 동사의 음성합성카드를 통해 발음되도록 되어 있다. MS-DOS

상의 각 어플리케이션이 화면에 표시하는 문자를 리얼타임으로 낭독할 수 있는 것외에 화면의 임의의 위치의 문자열을 마우스로 지정하여 낭독한다든지 화면의 문자를 확대 표시한다든지도 할 수 있다.

「030다이렉트슬롯」이라고 불리우는 확장슬롯을 1개 갖추고 있다. 이 슬롯에 비디오보드를 장착하면 외부에 컬러디스플레이를 접속할 수 있다. 1.44M바이트 FDD는 Mac Ⅱx에서 채용한 「수퍼드라이브」이며 현행의 400/800K바이트 타입이나 MS-DOS상의 데이터를 리드/라이트가 가능한 것이 특징이다.

애플사

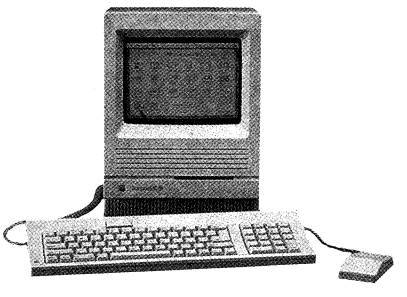
68030탑재의 매킨토시 SE/30 3모델 발표

미국 애플사는 매킨토시 패밀리의 중견모델 「매킨토시 SE」의 상위기종으로서 CPU에 68030을 채용한 「매킨토시 SE/30」 3모델을 개발, 3월부터 판매를 개시하였다.

3모델의 내역은 ① 1.44M바이트의 3.5인치 FDD 1대를 내장한 「플로피디스크모델」, ② 1.44M바이트 FDD 1대와 40M바이트 HDD 1대를 내장한 「40SC 하드디스크모델」, ③ 1.44M바이트 FDD 1대와 80M바이트 HDD를 내장한 「80SC 하드디스크모델」 등이다.

이 기종들은 CPU에 클럭주파수 15.6672MHz의 68030, 코프로세서에 같은 클럭주파수의 68882를 탑재하여 고속화를 꾀하고 있다. 이 때문에 68000을 채용한 현행의 매킨토시 SE에 대해서 연산속도에서 약100배, 처리속도에서 약4배의 스피드업을 실현하고 있다. 또 메인기판에는 CPU보드나 메모리보드, 비디오보드 등을 접속할 수 있는

〔그림 1〕 매킨토시 SE/30의 외관



모토롤라사

히다찌를 연방재판소에 제소

미국 모토롤라사는 히다찌제작소의 8/16비트 마이크로프로세서 「H8」이 모토롤라사의 특허를 침해하고 있어서 그 판매는 불공정경쟁에 해당한다고 해서 히다찌제작소, 히다찌아메리카 일제산업아메리카 등 3사를 상대로 하여 판매금지를 요구하는 소송을 시카고 연방재판소에 제소하였다. 소송에 의하면 모토롤라사는 1986년 5월에 히다찌 와의 사이에 특허 라이센스공여협정을 체결, 모토롤라사제의 마이크로프로세서 6800/68000계의 일부 특허라이센스를 히타찌에 주었는데 히다찌는 협정에 위반하고 그 직후에 「제조 턴 어라운드 타입(ZTAT)」이라고 불리우는 신형 마이크로프로세서를 개발하여 지금까지 불공정경쟁이 계속되고 있다고 지적, 히다찌는 모토롤라사의 특허를 적어도 4건을 침해하였다고 말하고 있다. 또 모토롤라사는 히다찌가 1990년 말까지 ZTAT의 제조를 단계적으로 중지할 것에 동의하였음에도 불구하고 1987년 10월부터 다시 라이센스협정에 위반하는 움직임을 보였다고 말하고 있다.

DTP시장

레이아웃기능 충실한 소프트웨어 대량 출하

DTP는 포스트스크립트 탑재의 페이지 프린터와 페이지레이아웃 소프트웨어를 중심으로 한 "간이 출판시스템"으로서 탄생하였다. 기본적으로는 퍼스널컴퓨터를 핵심으로 한 시스템으로 구성되는 것이 일반적이다.

DTP의 탄생에 있어서 가장 중요한 점은 ① 포스트스크립트라고 하는 디바이스 전용의 제어언어를 탑재한 페이지프린터의 등장, ② 오퍼레이션환경으로서 마우스를 주체로 한 비주얼 인터페이스의 보급 등과 같은 2가지를 들 수 있다. 이들 요소가 없었으면 DTP는 햇빛을 보지 못했을지도 모른다.

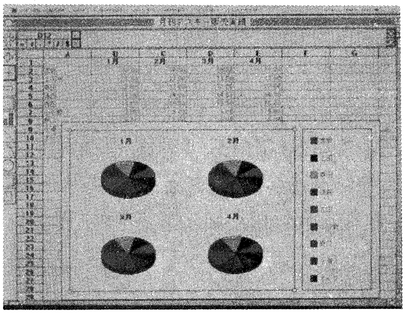
DTP라는 말은 당초 「수페이지정도의 도큐먼트를 보기쉽게 레이아웃하여 페이지프린터를 사용하여 아름답게 출력하여 설득력을 가지게 하는 시스템과 그 처리행정」이라는 의미로 사용되었다. 이러한 정의는 포스트스크립트를 핸들링하는 각종의 소프트웨어가 충실하게 됨에 따라서 본격적인 출판업무에 사용할 수 있는 시스템도 포함하도록 하는 식으로 발전하여 질적 양적인 변화를 이룩하여 왔다. 현재는 DTP라고 하면 어쨌든 「탁상출판」이나 「책상위출판」이라는 애매한 말로 파악되어 버려서 그 결과 "출판"이라는 말에 비중이 두어지는 경향이 지배적이다. 하지만 본래는 페이지단위의 도큐먼트작성법이라는 이미지로 시작되었다.

현재는 본격적인 인쇄시스템으로서 DTP를 이용하는 경우가 많다. 미국에

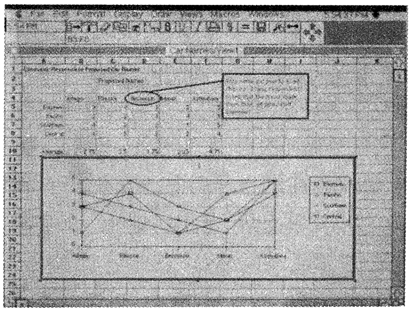
〔표 2〕 다양성 있는 IBMPC의 페이지 레이아웃 소프트웨어

상 품 명	Bylin	IMSI Publisher	Legend	Page Maker	Pages	PFS First Publisher	Publish-It	Ventura Publisher
회 사 명	Ashton-Tate	IMSI	NBI Inc.	Aldus Corp.	Pinpoint Published	Software Publishing Corp.	Timeworks Inc.	Xerox Corp.
버 전	1.0	1.0	1.1	3.0	1.02	2.0	1.12	2.0
가 격	295달러	495달러	695달러	795달러	79.95달러	129달러	199.95달러	895달러
포스트 스크립트 화일	×	×	×	○	×	×	×	○

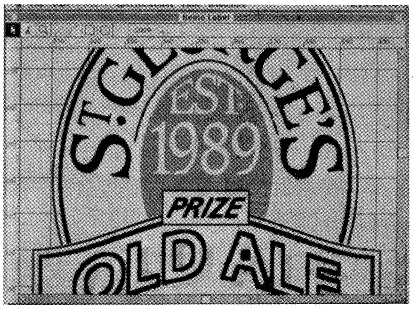
〔그림 2〕 각사의 통합형 스프레드 시트



Informix사의 통합형 스프레드시트 「Wingz」는 문장이나 그림, 그래프 등을 자유로이 레이아웃할 수 있다.



Ashton-Tate사의 통합형 스프레드시트 「Full Impact」도 레이아웃 중시의 설계사상을 유지하고 있다.



Letrase사의 「LetraStudio」는 벡터폰트를 편집, 가공할 수 있는 선진의 환경을 유지하고 있다.

서도 DTP라는 말의 중심시장은 퍼스널컴퓨터의 엔드유저가 대부분이다.

한편 DTP를 응용한 본격적인 출판비즈니스는 작년 전반까지 시대의 총아와 같이 미국출판업계에 크게 선전되어 왔지만 최근에는 당초의 흥분이 수습되어 가는 추세이며 그것보다는 그래픽스부분만을 처리한다든가 텍스트부분만을 처리한다고 하는 사정으로 부분적인 처리를 컴퓨터로 서포트하는 분업화 DTP가 많이 이용되고 있다.

이러한 경향은 최근 개개의 어플리케이션이 간단한 페이지 레이아웃기능을 장치하는 경우에 도움이 된다. 워드프로세서가 그렇고, 스프레드시트가 그렇고, 그래픽스가 그렇다. 그 배경에서는 페이지프린터의 보급에 따라서 1페이지전체의 이미지출력이라는 것이 일반적으로 되어 왔다고 하는 점이다. 포스트스크립트를 탑재한 페이지프린터의 보급이 없으면 레이아웃을 한다고 하는 의식은 그렇게 발달되지 않았을 것이다.

그러나 그것보다도 더욱 중요한 점은 역시 DTP자체에 대한 관심이 고조되고 있는 점이다. 분량이 많은 서적이나 잡지의 작성작업 모두를 DTP를 처리하고자 하는 의도는 현재 퍼스널컴퓨터의 처리능력에 비해서 부담이 매우 크다. 하지만 1매의 리포트에서 수백페이지의 잡지나 서적을 같은 레벨로 레이아웃할 수 있는 소프트웨어보다도 일상으로 사용하는 각 소프트웨어의 기능에 간편한 레이아웃기능을 부가하는 것으로 보다 실천적인 이용을 할 수 있도록 하는 생각은 지극히 당연한 생각이다. 각 소프트웨어가 DTP적인 수법을 도입하여 단일치에서도 페이지프린터의 기능을 인출하도록 하는 방향성은 최근들어 서서히 대두되고 있다.

어느 쪽이든 페이지레이아웃 소프트웨어가 없으면 DTP로는 안된다고 하는 인식이 없어지고 있는 것은 사실이다. 반대로 DTP라는 말은 더욱 하이엔드의 출판시장을 가리키는 것으로서 손이 닿지 않은 세계로 도약하고자 하는 것인지도 모른다.

미국에서의 DTP는 기반정비의 제1단계를 끝내고 개개의 소프트분야에서 레이아웃기능의 부가라는 확장을 향해서 착실히 진행하고 있다. 동시에 그래픽스화일의 세이브포맷의 통일화나 압축화일의 세이브포맷의 표준화 등의 작업도 진행되고 있으며, 각 소프트웨어로 작성한 레이아웃자체를 레이아웃소프트웨어에서 도입하고자 하는 움직임도 나타나고 있다.

따라서 앞으로 DTP가 흘러갈 방향은 무궁무진하다고 말할 수 있다. 아뭏튼 중요한 사실은 유저입장에서 사용하기 편리하도록 개선되어 간다는 점에는 틀림이 없다고 보아야 할 것이다.

미, JPI
OS/2용 Modula-2 발표

미국 Jensen and Partners International (JPI)사는 OS/2쪽의 Modula-2처리계「Top Speed Modula-2」를 1989년 2월부터 출하하기 시작하였다. 1패스의 고속 컴파일러에서 WordStar식 코맨드나 풀다운 메뉴 등의 프로그래밍환경을 특징으로 하고 있는 동처리계는 프레젠테이션 매니저의 어플리케이션 인터페이스와 OS/2 펑션콜을 모두 지원한다. 라이브러리 루틴은 MS-DOS판의 Modula-2와 완전히 호환성이 있으며 링커는 마이크로소프트사가 아니고 볼랜드사의 링커를 포함하며 이것에 의해 링크속도가 향상하였다고 한다.

미국 Hamilton
C셀을 OS/2에 이식

미국 Hamilton Laboratories사는 UNIX 4.3BSD판의 C Shell을 OS/2에 이식, 이를 「Hamilton C Shell」 명칭으로 발표하였다. 주된 특징은 ① 변수, 배열, 연산자 및 함수 등의 프로그램 구성요소를 완전히 네스트할 수 있으며 ② 과거의 코맨드이력을 기록하여 재이용할 수 있다는 점 ③ 입출력 리다이렉션이나 파이프 백그라운드 실장 등을 개선, ④ 정현표현에 의한 와일드 카드를 서포트 ⑤ 코맨드치환(별명정의)이 가능하다는 등이다.

마이크로소프트
C언어 소스코드 시판

마이크로소프트는 동사의 C언어컴파일러 Microsoft C5.1과 Quick C1.1의 런타임 라이브러리의 소스 코드(C와 어셈블리언어로 기술)를 발표하였다. 가격은 약 15만원정도이다.

미국 UNIFY
타사 DB에서도 이용 가능한 제4세대언어 발표

미국 UNIFY사는 제4세대언어 ACCELL 의 확장판「ACCELL/SQL」과 ANSI 표준 SQL의 레벨 Ⅱ에 준거한 데이터베이스관리 소프트웨어인「UNIFY2000」을 발표하였다.

UNIX상에서 동작하는 ACCELL/SQL의 특징은 타사의 데이터베이스관리 소프트웨어에서도 이용할 수 있다는 점이다. 또한 데이터베이스 엔진에 대한 인터페이스부를 분리독립시켜 이 인터페이스부를 바꾸어 쓰는 것으로 각사의 데이터베이스 관리 소프트웨어에 대응이 가능하다. 이것에 의해서 제4세대언어의 사실상의 표준으로 되는 것을 목표로 하고 있다. 현재 이용가능한 데이터베이스관리 소프트웨어는 UNIFY2000, Sybase사의 SQL Server, SCO사의 SCO/Integra 등이다.

헌터시스템
유닉스에서 사용 가능한 XDOS 개발

미국 Hunter System는 UNIX System V 혹은 4.3BSD에서 MS-DOS어플리케이션을 이용할 수 있도록 하는 XDOS를 개발하였다. 이것은 80386을 탑재한 UNIX머신에서 DOS어플리케이션을 직접 실행하는 VP/ix나 Merge386 등의 제품과는 다르며, 8086계 CPU용의 오브젝트 코드를 68030 등 아키텍처가 다른 CPU의 오브젝트 코드로 변환한다.

이미 68020/30판이 완성되어 모토롤라 88000, SPARC, MIPS의 각 RISC 프로세서판을 개발중이다.

소니가 미국에서 공급중인 NEWS에 X-DOS의 채용을 결정한 외에 모토롤라도 자사가 공급하는 UNIX System V에 XDOS를 공급한다고 공표하고 있다.

XDOS의 핵심은 바이너리 컴파일러이라는 점이다. 이것은 8086용 오브젝트 코드중의 명령을 타겟CPU의 명령으로 직접 변환한다. 예를 들면 68020의 경우 8086의 명령이 1.3~1.5명령으로 변환된다고 한다. 변환은 UNIX 머신상에서 한다.

컴파일시에는 키화일(key file)을 참조한다. 키화일은 프로그램의 광역프로나 자원이용의 패턴 등을 바이너리 컴파일러에 가리켜 하드웨어로 곧바로 접촉하고 있는 "순서가 나쁘다" 프로그램을 순서좋게 변신시키는 정보도 포함된다. 키화일은 어플리케이션마다 마련할 필요가 있다.

이미 123, dBASE Ⅲ PLUS, WordStar, Word Perfect, R:BASE 등이 68020/30머신의 UNIX하에서 동작하고 있다. 앞으로 Paradox, Framework, RageMaker 등도 동작하도록 할 예정이다.

변환부의 프로그램은 런타임 라이브러리를 실행시에 참조하며, DOS의 시스템콜이나 BIOS콜을 UNIX(혹은 윈도우 시스템 등)의 시스템 콜을 사용하여 에뮬레이트하는 것도 가능하다.

Hunter사에 의하면 타겟 시스템쪽으로 DOS 어플리케이션을 이식하였을 경우에 비해서 XDOS에서는 7~9할 정도의 성능을 발휘할 수 있다고 한다.

사칙연산

박승제/여주컴퓨터학원장

지난 호에서는 데이터 세그먼트를 설정하는 방법과 DS레지스터를 설정하는 방법에 대해 설명하였다. 이번 호부터는 연산명령에 대해서 설명한다.

덧셈 명령

[형식]

ADD 〈제1 오퍼랜드〉, 〈제2 오퍼랜드〉

[기능]

〈제1 오퍼랜드〉와 〈제2 오퍼랜드〉를 더하여 그 결과를 〈제1 오퍼랜드〉에 저장한다.

ADD는 Addition의 약어로 '더하라'라는 의미인데 이 명령에 의해 2개의 오퍼랜드 값을 더해서 앞의 오퍼랜드에 저장한다. ADD 명령을 사용할 수 있는 오퍼랜드의 종류는 다음과 같다.

① 범용 레지스터(8 / 16비트)와 범용 레지스터(8 / 16비트)
② 범용 레지스터와 메모리내의 데이터
③ 범용 레지스터와 수치
④ 메모리의 데이터와 범용 레지스터
⑤ 메모리의 데이터와 수치

여기서 범용 레지스터라고 함은 AX, BX, CX, DX, SI, DI, BP, SP를 말한다. 8088 CPU는 메모리와 메모리와의 직접적인 연산은 허용하지 않고 있으므로 이 경우에는 레지스터를 경유하여 연산을 하게 된다. 또 레지스터와 레지스터, 혹은 레지스터와 메모리 사이의 연산에서는 양쪽 다 자료의 크기가 일치해야 한다. 그러므로 필요시 자료의 형을 지정하는 의사 명령어를 기술해야 한다.

자료의 형을 지정하는 것은 메모리에 있는 데이터와 수치간의 연산을 할 경우이다. 예를 들어 BX에 1000H가 지정되어 있고 1000번지에는 1234H가 저장되어 있다고 하자. 이 경우에는 지난호에서 설명한대로 8088에서는 자료가 역워드 형식으로 저장되기 때문에 1000H번지에 하위 바이트인 34H가, 1001H에는 상위 바이트인 12H가 저장되게 된다 (그림 1 참조). 그렇기 때문에 다음과 같은 연산의 경우는 생각하기에 따라 연산결과가 달라지게 된다.

```
ADD [BX] , 0F4H
```

위의 명령은 'BX레지스터가 가리키는 메모리의 값'에 0F4H를 더하라는 의미이다. 이때 [BX]의 자료가 바이트(8비트) 크기인 34H를 가리키는 것인지, 워드(16비트) 크기인 1234H를 가리키는 것인지 알 수가 없게 된다. 만약 34H를 가리키는 경우라면 34H+0F4H 로 처리하므로 1000H의 자료는 28H가 되고 '자리올림'(이것을 캐리라고 부른다)이 발생하게 된다.

어셈블리에서는 8비트 단위로 연산을 한 후에는 ADD 명령으로 캐리가 발생하면 캐리가 발생했다는 신호만 낼 뿐 그것을 그 윗자리에 자동으로 더해주지는 않는다. 그러므로 1000H의 내용은 28H가 되고, 1001H의 값은 변하지 않게 된다.

[BX]의 자료가 워드를 나타내는 경우라면 1234H+0F4H로 처리하므로 하위바이트에서 생긴 캐리를 상위바이트인 12H에 더해 주므로 1000H의 값은28H가, 1001H의 값은 13H가 된다.

어셈블리에서는 데이터의 형을 지정하는데 있어 'PTR'(Pointer)이라는 의사명령을 사용한다. 위의 경우에 바이트 단위의 연산이라면

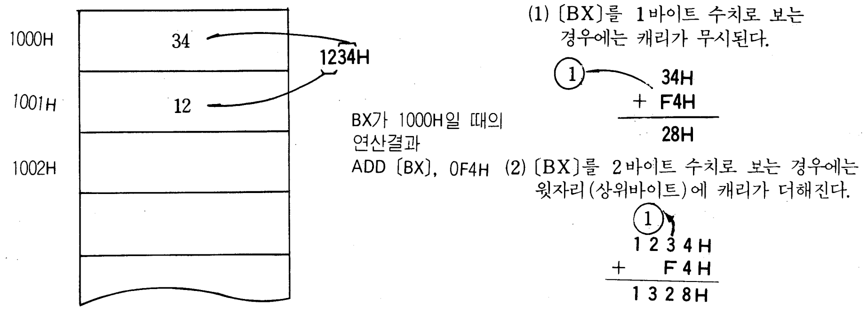
```
ADD BYTE PTR [BX] , 0F4H
```

워드 단위의 연산이라면

```
ADD WORD PTR [BX] , 0F4H
```

라고 기술해야 한다. 그러면 간단한 예제 프로그램을 통하여 ADD 명령의 사용법을 익혀보기로 한다.

〔그림 1〕 역워드 형식



〈예제 1〉
메모리의 특정한 곳에 1234H를 저장시킨 다음 그 값과 4142H를 더하여 그 결과를 화면에 출력하라.

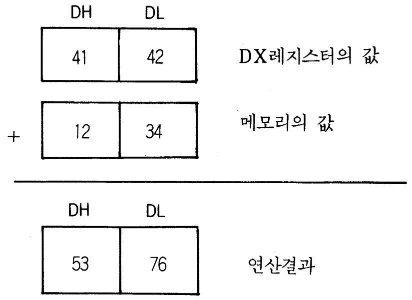
〈처리 방법〉
1234H는 지난 호에서 설명한 'DW' 의사명령을 이용하여 저장시킨다. 4142H는 DX레지스터에 기억시킨 다음 DX 레지스터를 제1 오퍼랜드로 하여 연산을 하도록 한다. 다음에는 DX의 값을 상위와 하위로 나누어 이 값을 아스키 코드로 하는 2개의 문자를 화면에 출력한다.

연산의 결과를 직접 화면에 출력하지 않고 아스키문자를 출력하는 이유는 DOS의 시스템 콜에서는 수치를 직접 출력하는 기능이 없기 때문이다. 수치를 화면에 직접 출력하는 것은 명령어를 조금 더 배워야만 가능하므로 우선은 그 값에 해당하는 아스키문자를 출력하는 것으로 한다. 위의 연산을 실행하는 과정 및 결과는 **그림 2**와 같다.

DL의 값과 DH의 값을 차례로 출력하면 'vS'(아스키 코드 76H와 53H)가 출력된다.

리스트 1에서 0007H를 보면 ADD DX, WORD PTR DATA' 처럼 DATA의 위치에 저장된 자료의 형을 명시하는 것이 있는데 여기서는 'WORD PTR'을 생략해도 된다. 왜냐하면 DATA는 'DW'에 의해 워드 형으로 지정이 되어 있기 때문이다. 그러나 DATA가 'DB'로 지정이 되었다면 'WORD PTR'를 생략하면 안된다.

〔그림 2〕 예제 1의 연산실행



●플래그 레지스터

CPU에는 플래그 레지스터(Flag Register) 라고 하는 특수한 용도의 레지스터가 있다. 이 레지스터는 CPU의 상태를 기억하는 것으로 16비트의 크기인데 각 비트명칭은 **그림 3**과 같 다.

●캐리 플래그: 연산의 결과 최상위 비트에서 그 윗자리로 자리올림(Carry)이 발생하거나 윗자리에서 빌어오기(Borrow)가 발생하면 1이 되고 그 이외에는 0이 된다.

●패리티 플래그: 연산의 결과 비트가 1의 값을 갖는 갯수가 짝수개이면 1이 되고 홀수개이면 0이 된다.

●하프 캐리 플래그: 4비트 연산시 하위 4비트에서 상위 4 비트로 자리올리기가 발생한 경우에 1이 된다. 이것은 수치를 BCD(2진화 10진수)로 연산할 경우에 쓰인다.

●제로 플래그: 연산의 결과가 0이면 1이 되고 그 외에는 0이 된다.

●사인 플래그: 연산의 결과 최상위 비트가 1이 되었을 때 1이 되고, 그 이외에는 0이 된다. 이것은 수치를 부호가 붙은 것으로 간주할 경우 음수가 되면 1이 되는 셈이다.

●오버플로우 플래그: 부호를 고려하여 연산을 했을 경우 오버플로우(Over Flow) 가 발생하는 경우에 1이 되며 그 외에는 0이 된다.

●디렉션 플래그: 이 플래그가 0인 경우에는 번지를 나타내는 레지스터의 값이 자동적으로 1씩 증가하고 1인 경우에는 반대로 1씩 감소한다.

●트랩 플래그: 이 플래그가 0이면 CPU는 정상적으로 프로그램을 수행한다. 1이면 한 명령을 수행할 때마다 내부 인터럽트를 발생하고 인터럽트 처리루틴을 수행한다. 이 플래그는 프로그램을 한 스텝씩 실행시킴으로써 프로그램의 실행과정을 추적하는데 쓰인다.

●인터럽트 플래그: 이 플래그가 0이면 외부 인터럽트는 무시되고 1이면 외부 인터럽트는 허용된다.

위의 플래그 중 앞에서 여섯개까지는 연산의 결과에 의해 플래그가 0이 되거나 1이 되고 나머지는 프로그래머가 0 혹은 1로 지정하게 된다.

캐리를 포함한 덧셈

앞의 예제 1은 16비트끼리의 덧셈이었지만 상위 바이트에서는 캐리가 발생하지 않았으므로 연산의 결과는 2바이트(16비트)가 된다. 그러나 상위 바이트에서 캐리가 발생한 경우에는 3바

〔리스트 1〕 1234H+4142H의 연산

```
0000                        MAIN    SEGMENT
                                    ASSUME  CS:MAIN,DS:MAIN
0000  8C C8                         MOV     AX,CS
0002  8E D8                         MOV     DS,AX
0004  BA 4142                       MOV     DX,4142H
0007  03 16 0017 R                  ADD     DX,WORD PTR DATA
000B  B4 02                         MOV     AH,2
000D  CD 21                         INT     21H
000F  8A D6                         MOV     DL,DH
0011  CD 21                         INT     21H
0013  B4 4C                         MOV     AH,4CH
0015  CD 21                         INT     21H
0017  1234                  DATA    DW      1234H
0019                        MAIN    ENDS
                                    END
```

이트(24비트)가 될 것이다.

예를 들어 8877+9988H의 결과는 121FFH가 된다(그림 4 참조). 이런 경우에는 한개의 레지스터로는 24비트를 기억할 수 없으므로 2개의 레지스터를 연결해서 연산결과를 기억하도록 한다. 이렇게 한다면 32비트(4바이트)까지 기억할 수 있게 된다.

단, 8088에서는 한번에 32비트의 자료를 처리해주지는 못하므로 실제 연산을 하는 경우라면 16비트씩 2회에 걸쳐서 하게 된다. 이때 하위 16비트에서 캐리가 발생하는 경우에는 그 캐리를 윗자리에 더해주게 된다.

8088에는 윗자리에 캐리만을 더하는 명령은 없고 일반적인 연산수(오퍼랜드)와 캐리를 함께 더하는 명령(ADC)만 있다. 그러므로 캐리만을 더하려는 경우라면 0을 캐리와 함께 더하게 된다. 그러면 예제를 가지고 ADC 명령의 쓰임을 익히기로 한다.

〈예제 2〉

4바이트의 수치 12 34 66 77H와 23 AB CD EFH를 더한 결과를 상위 바이트부터 아스키 문자로 바꾸어 화면에 출력하는 프로그램을 작성하라.

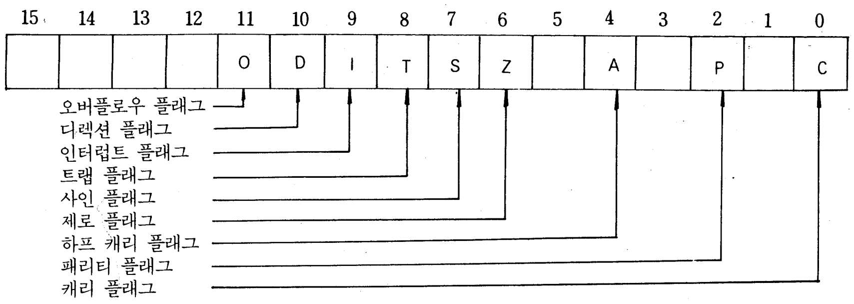
〈처리 방법〉

여러가지 방법이 있을 수가 있으나 여기서는 원래의 데이터는 메모리에 있는 것으로 한다. 32비트(4바이트 : 더블 워드)를 기억하기 위해 AX+BX에 12 34 66 77H를, CX+DX에 23 AB CD EFH를 저장하도록 한다.

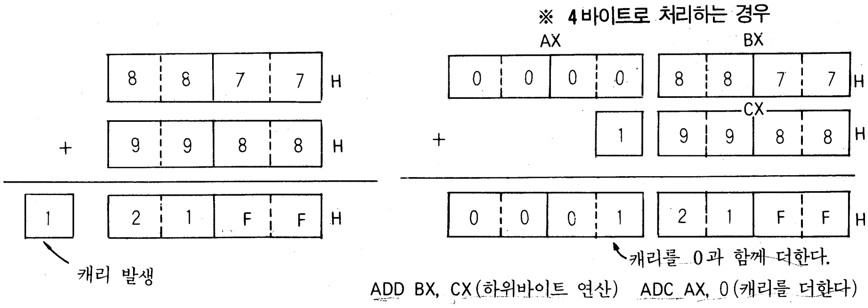
연산은 하위 워드인 BX와 DX를 먼저 더하고 상위 워드를 연산할 경우에는 캐리를 함께 더하도록 한다. 연산의 결과는 메모리에 저장했다가 그 끝에 '$'를 붙여 DOS기능 호출 9의 기능을 이용하여 문자열 형태로 출력하면 된다(리스트 2 참조).

1 워드(16비트)의 데이터가 메모리에 저장될 때는 상위 바이트와 하위 바이트가 바뀌어서 저장된다. 이것을 역워드 형식이라고 하는데 32비트의 자료를 4바이트 문자열로 취급하기 위해서는 이 자료들이 역워드 형식으로 저장되어서는 안될 것이다.

〔그림 3〕 플래그 레지스터



〔그림 4〕 2바이트 수치의 덧셈과 캐리 발생



그렇기 때문에 리스트 2에서는 AX와 BX를 각각 AH와 AL, BH와 BL로 나누어 한 바이트씩 메모리에 저장한다. 그러므로 'DATA5'는 'DB'로 지정되어 있다.

'MOV DATA5+1, AL'은 'DATA5'라는 레이블이 붙은 곳+1 번지에 AL 레지스터의 값을 저장하라는 의미이며, +2, +3… 등도 마찬가지로 해석하여야 한다.

지난 호에서도 설명을 했듯이 'DUP(데이터)'는 '데이터를 정한 수 만큼 메모리에 확보하라'라는 의미이다. 이때 데이터가 '?'일 경우에는 메모리의 영역만 확보할 뿐 특별히 초기화는 하지 않는다. 003FH의 의미는 5바이트를 확보하되 그 값은 아무 것이라도 괜찮다라는 것이다.

'MOV 〈레지스터〉, 〈레이블〉'은 '레이블이 붙은 번지의 내용'을 레지스터로 옮기는 것'을 의미하지만 'MOV 〈레지스터〉, OFFSET 〈레이블〉'은 레이블 자체의 번지를 레지스터에 기억시키는 것을 의미하게 된다.

'ADC'명령은 캐리와 함께 더하라는 명령인데 실제로는 캐리 플래그의 값을 더하게 된다. 캐리 플래그는 앞의 연산결과 자리올리기가 발생한 경우라면 '1', 발생하지 않았으면 '0'이 되므로 자릿수가 많은 수치를 연산할 경우, 처음의 덧셈은 'ADD'명령으로, 그 다음 자리수부터는 'ADC'명령으로 덧셈을 하면 된다.

위의 프로그램을 실행하면 화면에 '5α4f'(연산 결과는 35 E0 34 66H)가 출력된다.

●데이터를 교환하는 명령

8088 역워드 형식으로 저장되기 때문에 워드 형의 자료를 바이트 형의 자료로 치환하는 경우에는 위의 리스트 2와 같이 다소 번거로워진다. 8088에서는 이런 처리를 위해 'XCHG'라는 명령이 있다.

〔리스트 2〕 캐리를 포함한 덧셈

```
0000                    MAIN    SEGMENT
                                ASSUME  CS:MAIN,DS:MAIN
0000  8C C8                     MOV     AX,CS
0002  8E D8                     MOV     DS,AX
                                ;
0004  A1 0037 R                 MOV     AX, DATA1
0007  8B 1E 0039 R              MOV     BX, DATA2
000B  8B 0E 003B R              MOV     CX, DATA3
000F  8B 16 003D R              MOV     DX, DATA4
                                ;
0013  03 DA                     ADD     BX, DX
0015  13 C1                     ADC     AX, CX
0017  88 26 003F R              MOV     DATA5, AH
001B  A2 0040 R                 MOV     DATA5+1, AL
001E  88 3E 0041 R              MOV     DATA5+2, BH
0022  88 1E 0042 R              MOV     DATA5+3, BL
0026  C6 06 0043 R 24 90        MOV     DATA5+4, '$'
                                ;
002C  B4 09                     MOV     AH, 9
002E  BA 003F R                 MOV     DX, OFFSET DATA5
0031  CD 21                     INT     21H
                                ;
0033  B4 4C                     MOV     AH, 4CH
0035  CD 21                     INT     21H
0037  1234              DATA1   DW      1234H
0039  6677              DATA2   DW      6677H
003B  23AB              DATA3   DW      23ABH
003D  CDEF              DATA4   DW      0CDEFH
003F  0005[             DATA5   DB      5 DUP(?)
           ??

0044                    MAIN    ENDS
                                END
```

[형식]

XCHG 〈Op1〉, 〈Op2〉

[기능]

제1오퍼랜드와 제2오퍼랜드의 값을바꾼다.

〈Op1〉과 〈Op2〉로 지정할 수 있는 레지스터 및 메모리의 조합은 다음과 같다.

① 범용 레지스터(8 / 16비트)와 범용 레지스터(8 / 16비트)
② 범용 레지스터와 메모리내의 데이터
③ 메모리의 데이터와 범용 레지스터

XCHG 명령으로 교환하려는 자료의 크기는 일치해야 한다. XCHG 명령은 역워드 형식으로 저장된 데이터를 한 바이트씩 처리할 경우에 자주 쓰이는 명령이다.

리스트 2를 XCHG를 사용하여 프로그래밍하려면 0017H부터의 5행과 003F를 다음과 같이 바꾸면 된다.

```
0017  XCHG  AH, AL
 .    XCHG  BH, BL
 .    MOV   DATA5, AX
 .    MOV   DATA5+2, BX
 .    MOV   DATA5+4, '$'
 .
 .
003F  DATA5 DW 2 DUP(?)
```

뺄셈 명령

[형식]

SUB 〈Op1〉, 〈Op2〉

[기능]

〈Op1〉에서 〈Op2〉를 빼고 그 결과는 〈Op1〉에 저장한다.

오퍼랜드의 지정방법과 종류는 ADD 명령과 동일하다.

●캐리를 포함한 뺄셈

뺄셈의 경우 빼는 수가 더 커서 상위자리로부터 빌림(Borrow)이 발생하는 경우가 있다. 이 경우에도 캐리 플래그는 '1'로 세트된다.

SBB 명령은 여러 자리의 수차를 뺄셈할 경우에 캐리와 함께 뺄셈을 하는 명령이다. 덧셈의 경우와 마찬가지로 맨 하위바이트(혹은 워드)는 SUB 명령으로 뺄셈을 하고, 그 윗자리부터는 SBB 명령으로 뺄셈을 하면 된다. ADC 명령과 마찬가지로 캐리만을 빼는 경우는 없으므로 캐리만을 빼는 경우라면 캐리와 함께 0을 빼도록 한다.

〈예제 3〉

메모리에 3456 569AH를 기억시키고 이 값에서 F123H를 뺀 결과를 하위바이트부터 화면에 출력하라.

〈처리 방법〉

위의 수치를 필산으로 하면 그림 5와 같다. 8088에서는 워드(2바이트)의 수치를 직접 연산할 수 있으므로 먼저 하위워드부터 연산을 하게 된다.

789AH-F123H의 연산결과는 8777H가 되고, 윗자리에서 빌림(Borrow)이 발생하게 된다. 상위워드안 3456H는 뺄셈할 대상이 없으나 캐리가 발생하고 있으므로 3456H에서 캐리와 함께 0을 빼도록 한다.

출력은 리스트 2와 같이 계산 결과

를 메모리에 저장해두고 DOS 기능호출 9번의 '문자열 출력'루틴을 이용한다. 단, 위에서 하위바이트부터 차례대로 출력하라고 했는데 8088에서 워드의 자료는 역워드 형식으로 저장되므로 하위워드부터 메모리에 저장한다면 맨 하위바이트가 맨 처음에, 최상위 바이트가 맨 나중에 저장되게 된다(리스트 3 참조).

리스트 3의 프로그램은 코드 세그먼트와 데이터 세그먼트를 따로 설정하고 있다. 꼭 이렇게 해야만 되는 것은 아니지만 지난 호에서 데이터 세그먼트를 설정하는 방법을 배웠으므로 복습을 겸해 사용해본 것 뿐이다.

이 프로그램의 연산 결과로 DATA1에 저장된 값은 77 56 55 34H이므로 이 값에 '$를 붙여서 문자열로 취급하여 출력하면 'weU4'가 출력될 것이다.

● 음수의 표현방법

컴퓨터는 수치나 문자를 모두 2진수로 취급한다. 컴퓨터에서는 음수를 2의 보수에 의한 방법(2's Complement)으로 표현한다.

2의 보수에 의한 방법이란 1의 보수를 취한 값에 2진수 1을 더하는 것을 말하며, 1의 보수는 2진수의 각 비트를 반전, 즉 1을 0으로, 0을 1로 바꾸면 된다. 예를 들어 −87을 8비트로 표현하면 그림 6과 같다.

그러므로 8비트로 표현하는 경우에 −123은 $(10101001)_2$가 된다. 그런데 이 수치는 생각하기에 따라 +169로 볼 수도 있다. $(10101001)_2$를 양수로 보느냐 음수로 보느냐는 인간이 생각하기에 따라 전혀 다른 수치가 되지만 2진수의 표현은 둘 다 같은 결과가 된다.

그렇기 때문에 그 수치를 부호가 붙은 것으로 보는 경우에는 최상위 비트가 0이면 양수, 1이면 음수로 취급하기로 되어 있다. 그러므로 $(00000000)_2$~$(01111111)_2$는 양수로, $(10000000)_2$~$(11111111)_2$는 음수로 취급하게 된다. 이것은 16진수로 표현하면 양수는 0H~7FH까지, 음수인 경우에는 −1H~−80H까지이다(편의상 0은 양수쪽에 속해 있다.). 16비트인 경우에도 마찬

〔리스트 3〕 뺄셈 결과를 하위바이트로부터 출력

```
0000                           MAIN     SEGMENT
                                        ASSUME  CS:MAIN, DS:DATA
0000  B8 ---- R                         MOV     AX, DATA
0003  8E D8                             MOV     DS, AX
                                        ;
0005  B8 3456                           MOV     AX, 3456H
0008  BB 569A                           MOV     BX, 569AH
000B  B9 F123                           MOV     CX, 0F123H
                                        ;
000E  2B D9                             SUB     BX, CX
0010  1D 0000                           SBB     AX, 0
                                        ;
0013  89 1E 0000 R                      MOV     DATA1, BX
0017  A3 0002 R                         MOV     DATA1+2, AX
001A  C7 06 0004 R 0024                 MOV     DATA1+4, '$'
                                        ;
0020  B4 09                             MOV     AH, 9
0022  BA 0000 R                         MOV     DX, OFFSET DATA1
0025  CD 21                             INT     21H
                                        ;
0027  B4 4C                             MOV     AH, 4CH
0029  CD 21                             INT     21H
002B                           MAIN     ENDS
0000                           DATA     SEGMENT
0000  0003[                     DATA1   DW  3 DUP(?)
          ????
                    ]

0006                           DATA     ENDS
                                        END
```

〔그림 5〕 캐리를 포함한 뺄셈

	34	56	56	9A	H
−			F1	23	H
	34	55	65	77	H

〔그림 6〕 보수 표현 예

7	6	5	4	3	2	1	0	
0	1	0	1	0	1	1	1	+87
1	0	1	0	1	0	0	0	비트를 반전시킨다.
1	0	1	0	1	0	0	1	1을 더한다(−87).

가지로 0H~7FFFH까지가 양수, 8000H ~FFFFH까지를 음수로 취급하게 된다.

그런데 연산을 하다보면 연산의 결과가 이 범위를 넘어버리는 경우가 발생하게 된다. 이를테면 양수 50H와 40H를 더하면 90H가 되는데 이것은 위에서 설명한 음수의 범위에 속하게 된다. 그런데 이들의 자료가 2바이트 수치인 경우에는 0050H+0040H=0090H가 되어 양수의 영역에 속하게 된다.

이렇듯 같은 수치라 하더라도 그 수치를 표현하는 바이트 수에 따라서 양수의 영역에 속할 수도 있고, 음수의 영역에 속할 수도 있게 된다.

이러한 모순을 없애기 위해서 컴퓨터는 '워드'(Word)라는 개념을 도입하였다. 여기서 워드는 '단어'가 아니라 '단위'로 해석하는 것이 좋다. 워드란 컴퓨터가 한 번에 연산할 수 있는 자료의 바이트수를 나타내는 것인데 8비트나 16비트 컴퓨터에서는 1워드가 2바이트로 구성된다. 그러나 대형 기종에서는 워드가 4바이트로 구성된다. 위의 예에서 50H를 워드로 표현하면 0050H가 된다.

그러므로 바이트의 수치는 워드로, 워드의 수치는 더블 워드(4바이트)로 확장하여 연산을 한다면 위에서와 같은 모순은 생기지 않게 된다.

부호 확장명령

8088에는 바이트의 수치를 워드로, 워드의 수치를 더블워드로 변환하는 명령이 있다. 단, 모든 레지스터에 대하여 이런 명령을 적용하는 것은 아니고, AL 레지스터의 값을 AX로 변환한다거나, AX의 자료를 DX와 AX를 연결한 4바이트 자료로 확장할 경우에만 쓸 수 있다.

CBW나 CWD는 오퍼랜드를 갖지 않으며, 그 자체가 하나의 명령어가 된다. 참고로 말하자면 CBW는 Convert Byte to Word의 약어이며, CWD는 Convert Word to Double Word의 약어이다.

AL의 값을 확장할 경우에는 그수의 최상위 비트가 0이면 AH에 00H를 대입하지만 1이면 FFH를 대입하게 된다.

AX의 값을 확장할 경우에도 최상위 비트의 값에 의해 DX의 값이 0000H가 되거나 FFFFH가 된다. 그러면 예제를 통해 위의 명령어를 익혀보기로 하자.

〔리스트 4〕 부호 확장 명령을 이용한 연산

```
0000                            MAIN    SEGMENT
                                        ASSUME  CS:MAIN, DS:MAIN
0000  8C C8                             MOV     AX, CS
0002  8E D8                             MOV     DS, AX
                                        ;
0004  A0 0033 R                         MOV     AL, D1
0007  98                                CBW
0008  8B D8                             MOV     BX, AX
                                        ;
000A  A0 0034 R                         MOV     AL, D2
000D  98                                CBW
000E  93                                XCHG    AX, BX
                                        ;
000F  2B C3                             SUB     AX, BX
0011  A3 0037 R                         MOV     ANS, AX
                                        ;
0014  A0 0035 R                         MOV     AL, D3
0017  98                                CBW
0018  8B C8                             MOV     CX, AX
                                        ;
001A  A0 0036 R                         MOV     AL, D4
001D  98                                CBW
                                        ;
001E  03 C1                             ADD     AX, CX
0020  A3 0039 R                         MOV     ANS+2, AX
0023  C6 06 003B R 24                   MOV     BYTE PTR ANS+4, '$'
                                        ;
0028  B4 09                             MOV     AH,9
002A  BA 0037 R                         MOV     DX, OFFSET ANS
002D  CD 21                             INT     21H
                                        ;
002F  B4 4C                             MOV     AH,4CH
0031  CD 21                             INT     21H
0033  30                        D1      DB      30H
0034  50                        D2      DB      50H
0035  C0                        D3      DB      -40H
0036  A0                        D4      DB      -60H
0037  0002[                     ANS     DW  2  DUP(?)
          ????
                      ]

003B                            MAIN    ENDS
                                        END
```

〈예제 4〉

메모리의 적당한 곳에 +30H, +50H, −40H, −60H를 저장한 다음 양수끼리는 서로 뺄셈을 하고, 음수끼리는 서로 덧셈을 한 결과를 메모리에 저장한 뒤 그 결과를 화면에 출력하라.

〈처리 방법〉

−40H와 −60H는 소스 화일상에 기술하여 두면 어셈블러가 자동으로 치환하여 주므로 우리가 일부러 2의 보수를 취할 필요는 없다. 이 수치들은 부호가 붙어있으므로 부호 확장명령으로 2바이트 수치로 치환한 다음 연산을 하게 된다(**리스트 4** 참조).

〈프로그램 설명〉

0000H~0003H : 데이터 세그먼트의 설정

0004H~0009H : D1의 자료를 확장하여 BX에 보관

000AH~000EH : D2의 자료를 확장한 후 BX의 값과 AX의 값을 바꾼다.

AX는 0030H, BX는 0050H가 된다. AX와 BX의 값을 서로 바꾸는 이유는 AX에서 BX를 빼고 싶기 때문이다. BX에서 AX를 빼는 것으로 한다면 서로 바꿀 필요는 없다.

000FH~0013H : AX에서 BX를 뺀 다음 이 값을 'ANS'에 저장한다.

0014H~0019H : D3의 값을 확장한 후 CX에 저장한다.

001AH~0027H : D4의 자료를 확장하여 CX와 더한다. AX와 CX를더하고 있으므로 실제로는 −60H와 −40H를 더하게 된다(덧셈의 경우는 순서가 바뀌어도 괜찮다). 더한 결과는 ASN+2 번지에 저장된다. 문자열의 끝을 나타내는 '$'를 ANS+4에 저장한다.

0028H~002EH : ANS에 저장된 문자열을 화면에 출력한다.

002FH~003EH : 프로그램의 종료

0033H이후를 보면 음수의 경우에는 모두 2의 보수로 치환이 되어 저장된 것을 불 수 있다. 위의 프로그램을 필산으로 하면 **그림 7**과 같다.

위의 데이터는 역워드 형식으로 저장되므로 메모리에는 E0 FF 60 FF 24 순으로 저장되게 된다. 끝에 24H가 붙은 것은 '$'문자를 나중에 저장하기 때문이다.

〔그림 7〕 음수의 연산

```
     0030H                          −40H
−    0050H                     +    −60H
──────────                     ──────────
−     20H ───→ FFE0H               −A0H ───→ FF60H
```

그런데 이들 문자의 아스키 코드표를 보면 E0H는 α, 60H는 작은 따옴표를 반대로 해둔 문자가 할당되어 있고, FFH에는 문자가 할당되어있지 않다. 그러므로 리스트 4를 실행시키면 화면에는 위의 2문자만 출력될 것이다.

이 프로그램의 실행결과를 확인하기 위해서는 DOS 디스크에 들어있는 DEBUG. COM이 필요하다. DEBUG. COM으로 실행결과를 확인하는 과정을 화면 1에 표시했으니 참고하기 바란다.

결어

이번 호에서는 덧셈과 뺄셈 명령, 자료의 값을 교환하는 명령과 부호 확장 명령에 대해 알아보았다. 고급언어의 경우에는 연산자에 의해 자동적으로 처리되는 연산의 경우에도 어셈블리에서는 프로그래머가 일일이 자료의 크기와 부호를 고려하여 연산하여야 한다.

어셈블리에서 복잡한 산술연산을 할 경우는 별로 없겠고, 꼭 복잡한 연산을 해야 할 경우라면 수치연산 부분은 파스칼 등의 고급언어로 작성하여 두고 그 부분을 하나의 서브루틴으로 호출하여 쓰면 되겠지만 간단한 연산의 경우는 스스로 작성할 줄 알아야 한다. 연산 명령을 할 줄 알아야 다음에 설명할 조건에 의한 처리가 가능하기 때문이다.

〔화면 1〕 DEBUG.COM에 의한 실행결과 확인

```
C:\MASM>DEBUG LIST-4.EXE ↵  ←── 디버거를 기동하여 LIST-4.EXE를 로드한다.
-R ↵  ←── 레지스터를 표시하라는 디버거의 명령어
                                          LIST-4.EXE의 크기
AX=0000  BX=0000  CX=003B  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=105D  ES=105D  SS=106D  CS=106D  IP=0000   NV UP EI PL NZ NA PO NC
106D:0000 8CC8          MOV     AX,CS     CS와 DS의 값이 다르다(CS와 DS의 값은
-U 0,3B ↵                                 컴퓨터의 상태에 따라 각각 다르다).
106D:0000 8CC8          MOV     AX,CS         0H부터 3BH까지 역어셈블한다.
106D:0002 8ED8          MOV     DS,AX
106D:0004 A03300        MOV     AL,[0033]
106D:0007 98            CBW
106D:0008 8BD8          MOV     BX,AX
106D:000A A03400        MOV     AL,[0034]
106D:000D 98            CBW
106D:000E 93            XCHG    BX,AX
106D:000F 2BC3          SUB     AX,BX
106D:0011 A33700        MOV     [0037],AX
106D:0014 A03500        MOV     AL,[0035]
106D:0017 98            CBW
106D:0018 8BC8          MOV     CX,AX
```

```
106D:001A A03600        MOV     AL,[0036]
106D:001D 98            CBW
106D:001E 03C1          ADD     AX,CX
106D:0020 A33900        MOV     [0039],AX
106D:0023 C6063B0024    MOV     BYTE PTR [003B],24
106D:0028 B409          MOV     AH,09
106D:002A BA3700        MOV     DX,0037
106D:002D CD21          INT     21
106D:002F B44C          MOV     AH,4C              ← 저장된 DATA
106D:0031 CD21          INT     21                 ← 연산결과가 저장될 영역
106D:0033 3050C0        XOR     [BX+SI-40],DL
106D:0036 A00000        MOV     AL,[0000]
106D:0039 0000          ADD     [BX+SI],AL         ← DATA가 저장된 위치로 이 내용은 무의미하다.
106D:003B 50            PUSH    AX
-G=0,31 ↵               ← 프로그램의 처음부터 31H까지 실행한다.
a₩                      ← 실행결과
AX=4C24  BX=0050  CX=FFC0  DX=0037  SP=0000  BP=0000  SI=0000  DI=0000
DS=106D  ES=105D  SS=106D  CS=106D  IP=0031   NV UP EI NG NZ NA PE CY
106D:0031 CD21          INT     21                 ← CS와 DS의 값이 일치되었다.
-D 33 ↵                 ← 33번지 이후를 화면에 덤프
106D:0030           30 50 C0 A0 E0-FF 60 FF 24 E8 AE 52 83     0P....'.$..R.
106D:0040  C4 04 FF 36 CE 5C B8 15-09 50 E8 A0 52 83 C4 04   ...6.\...P..R...
106D:0050  8A 46 04 98 50 B8 0D 39-50 E8 10 10 8B E5 5D C3   .F..P..9P.....].
106D:0060  55 8B EC 81 EC 00 10 C7-06 E6 5D 00 00 8D 86 00   U.........].....
106D:0070  F0 A3 00 54 8A 46 04 98-50 B8 A6 3A 50 E8 EC 0F   ...T.F..P..:P...
106D:0080  83 C4 04 80 7E 04 03 75-05 B8 18 09 EB 03 B8 20   ....~..u.......
106D:0090  09 A3 9E 13 FF 36 CE 5C-B8 27 09 50 E8 4E 52 83   .....6.\.'.P.NR.
106D:00A0  C4 04 FF 36 9E 13 B8 2A-09 50 FF 36 CE 5C E8 DD   ...6...*.P.6.\..
106D:00B0  41 83 C4                                          A..
-Q ↵                    ← 디버거를 종료

C:\MASM>
```

(덤프 주석)
- 30 50 C0 A0: 초기의 DATA
- E0-FF: FFE0H (상위와 하위가 바뀌어 있다.)
- 24: 문자열의 끝을 나타내는 '$' 문자의 아스키코드
- 60 FF: FF60H (상위와 하위가 바뀌어 있다.)

〈용어해설〉

DOS기능호출

초창기의 프로그래머들은 프로그램을 작성할 때 데이터의 입력과 처리과정은 물론 처리된 데이터를 출력하기 위한 주변장치를 제어하는 루틴까지도 모두 세세히 만들어 주어야 했다. 이는 전반적인 하드웨어 메카니즘을 모두 이해해야만 가능한 일이었기 때문에 컴퓨터 전문가가 아닌 사람이 프로그램을 작성하는 것은 어려운 일이었다.

쉽게 볼 수 있는 예로 문자 하나를 키보드로 받아들이거나 데이터를 디스크로 기억시키는 일, 심지어 화면에 문자 하나를 출력하는데 필요한 제어 프로그램 루틴들을 생각할 수 있다.

그 후 DOS의 기능이 확대되면서 이런 주변장치 제어루틴들을 표준적으로 작성해 DOS속에 기억시켜 놓은 후 각 기능별로 코드(번호)를 할당하여 필요로 할 때만 이 코드를 호출하여 해당 기능을 얻게 하였다. 이들 코드를 DOS기능 번호(DOS Function Number)라 부르며 프로그램 중에 이 코드를 호출하는 것을 DOS기능호출(DOS Function Call)이라고 하는 것이다.

이렇게 하면 주변장치를 제어해야 하는 곳에서 해당 코드만 호출하면 되므로 전문가가 아니더라도 쉽게 프로그램을 작성할 수 있는 것이다. 호출 방법을 보면 AH레지스터에 원하는 기능번호를 넣고 가로채기 명령 INT 21H를 실행시키면 된다.(데이터가 필요한 경우는 DX, 또는 CX를 이용)

예를 들면 알파벳 'A'를 화면에 나타나게 하기 위해서는 한 문자를 화면에 출력시키는 기능번호 '2'를 AH 레지스터에 기억시키고 (MOV DL, 65d), DL 레지스터에 원하는 문자의 ASCII코드를 기억시킨 후(MOV DL, 65d), INT 21H를 실행시키면 된다.

프로그래밍 규칙

박성현/디베이스 프로그래머

디베이스를 배우는데 있어서 한가지 명령어의 특징과 기능을 아는 것보다 어떤 때에 사용하는 것인지를 알아야 그 명령어를 확실히 알게 되는 것이다. 이 강좌 역시 명령어의 성격보다 어떤 때에 사용하고, 어떻게 사용하는가에 역점을 두고 진행을 하고 있으므로 반드시 디베이스 매뉴얼과 함께 보길 먼저 당부한다.

디베이스 프로그래밍은 지난호에서도 그 예를 보았듯이 문번호없이 차례로 진행해가는 구조적(Structured)형식으로 진행하며 그 방법에 있어서도 C나 파스칼과 비슷한 유형을 하고 있다. 어려운 부분은 모두 명령어 하나로 처리되어 그 사용이 매우 간단하다는 매력을 가지고 있어 누구든지 쉽게 접할 수 있는 것이다. 또한 도트 프롬프트 상에서의 명령과 완전히 일치한다는 장점이 있어 쉽게 익힐 수 있다.

지난호에서는 디베이스에서의 프로그램이란 도트 프롬프트 상에서의 명령을 순서대로 나열된 상태로 이 순서를 제어하기 위한 조건루프를 설명하였다.

이번호에서는 이러한 프로그램을 작성하는데 있어서의 기본적인 규칙을 알아보자.

기본적인 규칙

(1) SET TALK OFF

프롬프트상에서 SET TALK OFF 상태와 SET TALK ON 상태를 비교해 보자.

```
. set talk off <- 비대화식 모드로 지정한다.
. var1=10      <- var1이라는 변수에 숫자 10을 넣어라
. ? var1       <- var1라는 변수가 무엇이냐?
      10       <- 10입니다.
. set talk on  <- 대화식 모드로 지정한다.
. var2=20      <- var2이라는 변수에 숫자 20을 넣어라
      20       <- 20이라고 넣었읍니다.
. ? var2       <- var2라는 변수가 무엇이냐?
      20       <- 20입니다.
```

만약 프로그램에서 명령할 때마다 대화식으로 하면 화면의 출력을 하고자 할 때가 아닌데도 화면에 계속 응답을 하게 될 것이다. 친절히 대화를 하는 것이 아니라 이제는 말대꾸가 되어 버릴 것이다. 이러한 말대꾸를 하지 말라는 명령이 바로 SET TALK OFF 이다.

이 명령은 프로그램 메인부분의 첫머리에 반드시 있어야 하는 디베이스만의 특수명령이다.

그러나 디베이스 프로그램 실행중 에러가 발생했을 경우 이때는 SET TALK ON 상태에서 실행하여 에러가 어떻게 발생되고 있는지를 알아 보는데 사용할 수도 있다.

(2) DO 〈프로그램 화일이름〉

디베이스 프로그램을 실행시키고자 할 때는 이와 같은 방법을 사용한다. 프로그램내에서 다른 프로그램을 부를 때도 이와 같이 부른다. DO로 불리워진 프로그램의 실행이 끝나면 제어는 불렀던 프로그램으로 돌아가거나 도트 프롬프트에서 호출된 경우는 키보드 코드로 돌아간다.

(3) RETURN

프로그램이 끝나면 자동으로 프로그램화일에서 빠져나오게 된다. 그러나 프로그램의 중간에 빠져나오게 하는 명령이 필요한 경우가 생긴다. 만약에 어느 조건에 해당되면 프로그램을 끝내고 조건에 해당되지 않으면 다른 작업을 계속하게 할 경우에 RETURN 명령을 사용한다. 이때는 반드시 자기를 부른 프로그램으로 돌아간다.

다음 예를 통하여 살펴보자.

```
IF var1=20      <- 만약에 var1이라는 변수가 20이면
     RETURN     <- 나를 부른 프로그램으로 돌아간다.
ELSE            <- 아니면 다음을 수행
  .......
  .......
ENDIF
```

(4) NOTE, *

때때로 프로그램에 설명을 첨가해야 할 필요성이 있다. 즉, 명령어 라인이 "NOTE"로 시작되거나 "*"으로 시작되면 실행되지 않는 라인이 된다. 그러므로 이를 이용하면 프로그램내에 주석을 달 수 있게 된다.

(5) &&

NOTE나 *과 마찬가지로 주석을 달기 위한 것이나 실행되는 명령어뒤에 위치하여 실행명령의 설명을 달 수 있다.

(6) ;

프로그램을 작성하다보면 실행 명령 라인이 한줄에 끝나지 않고 다음 줄에

연속될 때는 첫번째 라인의 마지막 부분에 ';'라고 표시하여 명령이 계속된다는 것을 알린다.

(7) SET ECHO ON

디베이스 프로그램을 실행시킬 때 원하는 방식으로 실행되지 않을 경우, 어느 부분에서 잘못되어 있는가를 알아 볼 수 있도록 현재 실행되는 라인의 명령이 스크린에 나타나도록 하는 명령이다. (에러가 발생하였을 때에 사용하며 이때 SET DEBUG ON 명령과 함께 사용하면 스크린에 나오는 실행명령이 프린터에 나오게 된다. 스크린에서는 정상적으로 프로그램이 돌아가고 프린터에서는 실행되는 명령이 차례로 출력된다.)

데이터베이스 화일의 구조

다음 리스트 1을 통하여 간단하게 데이터베이스의 화일구조에 관하여 알아보자. 먼저 현재 사용하는 아무 데이터베이스 화일을 열어 놓고 리스트 1을 순서대로 입력해보자.

앞의 결과로 다음과 같이 표 1을 만들 수 있으며, BOF 또는 EOF가 아니면 SKIP하는 수가 초과되어도 에러가 발생하지 않고 초과되는 경우에는 BOF나 EOF로 간다. EOF는 마지막 레코드보다 레코드번호가 하나 증가하여 그 수치를 가지지만 BOF는 항상 첫번째 레코드와 같은 번호를 가지고 있음을 유의해야 한다. 여기서 아래의 간단한 리스트에서 중요한 사실을 알아낼 수 있다. 이 리스트는 모든 레코드의 지정한 필드만을 보기 위한 것이다.(편의상 앞에 번호를 붙인다.)

```
1  SET TALK OFF
2  GO TOP
3  DO WHILE .NOT. EOF()
4        ? NAME,TELNO
5        SKIP
6  ENDDO
7  SET TALK ON
```

〔리스트 1〕

```
. use pc03    <- 데이터베이스 화일을 연다.
. go top      <- 화일의 가장 꼭대기 레코드로 가라  (레코드 번호 1로 간다)
. ? recno()   <- 현재의 레코드 번호는?             (1번)
. ? bof()     <- 지금 위치가 화일의 시작부분이냐?  (아니오)
. skip -1     <- 한칸 위로 이동하라
. ? recno()   <- 현재의 레코드 번호는?             (1번)
. ? bof()     <- 지금 위치가 화일의 시작부분이냐?  (예)
. skip -1     <- 한칸 위로 이동하라                (에러 발생)
. go top      <- 다시 화일의 꼭대기 레코드로 가라  (레코드 번호 1로 간다)
. skip -10    <- 10칸 위로 이동하라.               (에러 발생 않는다)
. ? bof()     <- 지금 위치가 화일의 시작부분이냐?  (예)
. ? recno()   <- 현재의 레코드 번호는?             (1번)
. skip -1     <- 한칸 위로 이동하라                (에러 발생)
. go bottom   <- 화일의 마지막 레코드로 가라       (20 번으로 간다)
. ? recno()   <- 현재의 레코드 번호는?             (20 번)
. ? eof()     <- 현재의 위치가 화일의 끝이냐?      (아니요)
. skip        <- 한칸 아래로 이동하라
. ? recno()   <- 현재의 레코드 번호는?             (21 번)
. ? eof()     <- 현재의 위치가 화일의 끝이냐?      (예)
. skip        <- 한칸 아래로 이동하라              (에러 발생)
. go bottom   <- 다시 마지막 레코드로 가라
. skip 10     <- 10칸 뒤로 이동하라                (에러 발생되지 않는다)
. ? recno()   <- 현재의 레코드 번호는?             (21 번)
. skip        <- 한칸 아래로 이동하라              (에러 발생)
```

〔표 1〕

레코드 번호	20개의 레코드 존재
1	BOF( )=.T.
	TOP
2	
⋮	
⋮	
19	
20	BOTTOM
21	EOF( )=.T.

DO WHILE .NOT. EOF()라는 명령을 보자. 현재의 레코드가 화일의 끝이 아닌 동안 계속 작업하라는 명령으로 화일의 처음부터 차례로 보다가 끝을 내게 할 때 이 명령을 사용한다. 그러면 앞의 리스트를 분석해 보자.

1. 비대화식 모드로 지정한다.
2. 화일의 꼭대기에 있는 레코드로 이동한다.
3. 현재의 레코드가 화일의 끝이 아닌 동안 계속한다. 현재는 첫번째 레코드에 있으므로 EOF()=.F.가 되고 앞에 .NOT.이 있어 여기서의 조건은 .T.가 되어 DO WHILE...ENDDO LOOP 문 안으로 들어가게 된다.
4. 지정한 필드를 디스플레이 해준다.
5. 다음 레코드로 한칸 이동한다.
6. 루프문이 끝났으므로 루프의 처음으로 가게 한다.
7. 다시 3으로 가서 현재의 레코드가 화일의 끝인가 비교한다. 이때 화일의 끝에 와서 EOF()=.T.가 되어 바로 앞의 .NOT.과 함께 전체적으로 .F.가 되어 루프안으로 들어가지 못하고 곧바로 ENDDO 다음의 명령인 대화식 모드로 전환하게 된다.

여기서 우리가 발견해 낼 수 있는 것이 DO WHILE 다음의 조건이 논리적으로 계산하여 참인가 거짓인가를 계산한 뒤 참인 경우에는 루프안으로 들어가게 하고 거짓인 경우에는 루프

를 벗어나게 된다는 것이다. 그렇다면 항상 루프안으로 들어가게 하기 위해서는 DO WHILE .T.라고 명령하면 된다는 것을 깨달을 수가 있다. 이렇게 하면 영원히 계속 루프를 돌게 될 것이다. (프로그램 작성시 이러한 경우는 상당히 많이 쓴다.)

또한 프로그램내에서 반드시 실행되어서는 안될 부분은 라인의 처음에 '*'를 넣어 주석문으로 처리할 수도 있지만 다음과 같이 처리하면 조건이 항상 거짓상태가 되어 이 부분은 실행되지 않는다는 것을 알 수 있다.

```
IF .F.
   ...........
   ...........
ENDIF
```

데이터의 입력방법

프로그램 수행중 데이터를 입력하거나 프로그램의 실행을 제어하기 위한 입력을 어떻게 처리할 것인가는 다음과 같은 WAIT 〈메시지〉 TO var, ACCEPT 〈메시지〉 TO var, INPUT 〈메시지〉 TO var, @ yy, xx SAY... GET variable / READ 등의 여러가지 방법이 있다.

(1) WAIT

WAIT TO는 문자 한글자를 받을 때까지 계속 기다려 주는 명령이다. 만약 WAIT라고만 입력하면 디베이스에서는 Press any key to continue... 라는 메시지를 직접 보여준다. 이는 아무키나 눌러서 계속 진행되도록 하는데 유용하게 쓰인다.

만약 한글로 그 메시지를 바꾸어야 할 경우는 WAIT "아무키나 누르십시요", 또는 아무 표시도 필요가 없을 경우는 WAIT" "라고 하면 된다. 그리고 뒤에 'TO 변수이름'을 붙이면 아무키나 누른 그 내용이 그 변수에 저장된다. 이때의 변수는 처음나오는 변수든 사용하던 변수든 상관이 없다.

(2) ACCEPT

이 명령은 키보드입력을 할 수 있게 하기 위해 명령화일에서 중요하게 사용된다. 이 명령은 실행되는 시점에서 메시지를 표시하고 사용자의 데이터입력을 기다린다. 사용자가 적절한 데이터를 입력하고 리턴을 치면 입력한 데이터가 문자데이터 형태로 지정한 메모리 변수에 저장된다.

다음과 같이 ACCEPT "당신의 이름은?" TO mname하게 되면 '당신의 이름은?'하고 메시지가 나타나면 이름을 입력하고 ?mname을 입력하면 이름이 mname에 저장되는 것이다. 데이터의 입력은 리탄을 치면 끝이 난다. ACCEPT문으로 입력할 수 있는 최대 문자는 254자까지 가능하다.

(3) INPUT

ACCEPT와 똑같은 방법으로 사용한다. 단지 입력할 때 숫자를 입력하면 수치형 데이터로 저장되고 문자형으로 입력하면 문자형으로 저장된다. (반드시 앞뒤에 따옴표를 붙여야 한다. 만약 따옴표를 붙이지 않고 문자를 입력하면 그 문자를 변수로 간주하고 그런 변수가 없음을 알리는 에러메시지가 나올 것이다.)

다음과 같이 INPUT "당신의 나이는?" TO age하게 되면 나이를 묻는 메시지가 나타나고, 나이를 입력한 다음 ?age 하면 age에 나이가 저장된다.

논리형으로 입력할 때는 앞뒤에 "." 을 넣어서 .T.나 .F. 형식으로 입력하고 날짜형으로 입력할 때는 CTOD("04/10/89")라고 해야 입력이 가능하다.

앞의 3가지 방법은 모두 메시지를 현재의 가로줄에서 첫번째 세로칸부터 메시지를 표시해 주므로 화면의 중간에 메시지가 표시되기를 원할 때는 앞의 공백이 반드시 필요하다.

(4) @ yy, xx SAY... GET... READ

프로그램을 작성할 때 가장 많이 사용되는 데이터 입력방법으로 지정한 위치에서 입력을 받을 수 있으며, 항상 미리 정의된 변수만을 사용해야 하므로 입력하기 전의 내용을 볼 수가 있으며, 리턴을 침으로써 그 내용을 그대로 유지한채 입력할 수도 있다. 또 입력할 수 있는 형태들을 원하는 대로 조정할 수 있는 기능도 갖추고 있다.

여기서 SAY는 메시지를 보여 주는 것이고 GET은 변수값을 반전상태로 보여 주고 있다가 READ라는 명령에 의하여 화면에 나타나는 GET의 반전부분이 입력 대기상태로 들어 간다. 만약 READ가 없다면 디스플레이 상태로 끝나고 만다.

@ yy, xx SAY... GET...을 여러번 사용한 후 READ가 나오면 한꺼번에 입력을 할 수 있다.

다음의 예는 데이터를 제어할 때 흔히 사용하는 방법이다.

```
1  ans=[?]
2  DO WHILE .NOT. ans$[YN]
3     ans=[?]
4     @ 18,20 SAY "Yes or No" GET ans PICTURE [!]
5     READ
6  ENDDO
```

앞의 리스트를 분석해 보면 다음과 같다.

1. ans라는 변수값의 초기치를 지정해 주는 것이다.
2. ans변수의 값이 "Y" 또는 "N"이 아닌 동안 계속 루프를 실행하라는 명령이다. 초기치가 '?'이므로 반드시 실행한다.
3. 다시 초기치를 지정해 준다. (여기서는 루프를 돌아서 다시 돌아왔을 경우 처음에는 ans=[?]이어서 '?' 표시가 있었는데, 예를 들어 'A'자를 입력했을 경우 화면에 ans 값의 입력 대기란에 'A'자가 나오지 않고 '?'가

나오도록 하기위한 명령이다.)
4. 지정한 위치에 그 내용을 보여준다. PICTURE [!]는 소문자를 입력하여도 대문자로 받아 들이게 하는 명령이다. 만약 이를 생략한다면 앞의 2번라인을 DO WHILE .NOT. ans$ [YNyn]이라고 해야 될 것이다.
5. 앞에 나온 GET의 부분으로 커서를 옮겨서 입력대기하게 한다.
6. 매치되는 DO WHILE의 위치인 2번 라인으로 가게 한다.(디베이스의 루프는 반드시 처음으로 돌아가서 조건을 판단한 후에 루프문 안으로 들어가느냐 빠져 나가느냐를 결정하게 되어 있다. 그러므로 디베이스의 ENDDO 라는 명령은 아무런 역할없이 여기서 반복 구조가 끝이 났으니 처음으로 돌아가서 다시 반복문의 조건을 비교할 뿐이다 .)

상위프로그램과 하위프로그램

프로그램을 작성하다 보면 1개의 화일로 끝이 나지 않고 프로그램 화일이 또 다른 프로그램 화일을 호출하게 된다. 여기서 다른 프로그램을 부른 프로그램을 상위프로그램(모 프로그램)이라 하고, 불리우는 프로그램을 하위프로그램(자 프로그램)이라고 하자. 여기서는 메모리 변수와의 관계가 중요시 된다. 하지만 이렇게 생각하면 매우 간단하다. "엄마가 만든 것은 아들이 쓸 수 있지만 아들이 만든 것은 엄마가 쓸 수 없다"

상위프로그램에서 정의된 메모리 변수는 하위프로그램에서 사용이 가능하고 만약 하위프로그램에서 그 내용이 변한 후에 상위프로그램으로 다시 돌아왔을 때에도 그 내용은 하위프로그램에서 변경한 내용 그대로이다.

하지만 하위프로그램에서 새로이 정의한 변수는 상위프로그램으로 돌아갈 때 모두 없어져 버린다. 하위프로그램에서 사용될 변수는 미리 상위프로그램에서 정의를 해 놓든지 하위프로그램의 첫머리에 "다음의 변수명은 여기서 정의하게 되는 것이지만 모 프로그램으로 돌아가서도 계속 존재하라"는 명령을 주면 되는데 그 명령이 PUBLIC 〈변수 리스트〉이다. 여기서 PUBLIC은 메모리 변수를 전체적으로 정의하게 되는 것이다.

또 하위프로그램에서 "다음의 변수명은 상위프로그램에서 정의된 변수명이지만 여기서 사용하고 돌아갈 때는 원래의 변수값을 그대로 유지한채 돌아가라"는 명령을 화일의 첫머리에 넣어 두면 될 것이다. 이러한 명령으로 PRIVATE 〈변수 리스트〉하게 되면 현재의 프로그램이 완료되면 변수는 지워지고 반면에 먼저번에 정의된 변수로서의 위치를 갖게되는 것이다.

여기까지의 설명으로 디베이스의 개요부터 핵심적인 내용의 설명을 마쳤다. 그러나 아직까지 프로그램을 막상 작성하려면 어렵게 느껴지는 것이 대부분의 독자들이 느끼는 공통된 생각일 것이다. 따라서 지금까지 설명한 내용을 토대로 매뉴얼을 보면서 각종 명령어와 함수를 익혀 놓길 바란다.

프로그램을 가장 쉽게 배우는 방법은 아무리 많은 이론적 설명을 듣는 것보다 남이 작성한 프로그램을 분석해 보고 이를 토대로 고쳐본 후 다시 이와 비슷한 프로그램을 작성해 보는 것이 가장 쉽게 배우는 방법중의 하나일 것이다. 가능한 훌륭한 프로그래머가 작성한 프로그램을 분석하여 익히는 것이 더 효과적일 것이다.

다음호부터는 예제 프로그램을 완전히 분석하여 독자들이 그와 유사한 프로그램을 직접 작성해 볼 수 있도록 유도해 보고자 한다. 디베이스의 프로그래머가 될 수 있다는 자부심을 갖길 바라며… PC

레코드판 관리 프로그램

이기성 / 장왕교재연구원 상무

이번호의 예제로 든 레코드판 관리 프로그램은 200개의 레코드판에 수록된 곡명을 랜덤화일에 입력시키고 입력번호로 찾기, 레코드판 이름으로 찾기, 코드번호로 찾기, 연속하여 찾기와 같이 다양한 방법으로 데이터에 접근할 수 있는 프로그램이다.

원래 이 프로그램은 필자의 책인 '알기쉬운 BASIC프로그램 모음'에서 소개한 전화번호 관리프로그램을 조남훈씨가 레코드판 관리프로그램으로 수정한 것이다. 실제의 이 프로그램은 전화번호 관리프로그램에 있는 모든 루틴을 다 사용한 것이나, 여기에서는 베이직명령어를 익히면서 이제 서서히 프로그램을 작성해 보려는 독자들을 위해 몇개의 필수적인 루틴만 골라서 설명하고자 한다. 이 프로그램은 레코드판 관리는 물론 카세트테이프 관리도 가능하여 카세트 한개당 앞뒤로 10곡씩 20곡을 기록하는 실무프로그램으로도 쓰일 수 있다.

프로그램의 구성

레코드판 관리프로그램은 데이터 화일을 처음만들기, 데이터를 입력하기, 데이터를 찾아보기 등 3가지 큰 루틴으로 구성되어 있는데, 실제 업무 프로그램에서 메뉴처리와 루틴처리가 어떻게 쓰이는지를 살펴보기 바란다.

전반적인 프로그램 구성은 그림 1의 흐름도에 나타내었으며 시작화면에서 스트링변수 'OO'에 의해 각 루틴으로 분기시키고 있다. 참고로 분기명령은 지난호에서 설명한 ON.. GOTO..문을 이용하였음을 밝혀 둔다.

```
10 ON OO GOTO 110,1860,1580,990
20 ON CC GOTO 500,1030,2120,2230,710,710
```

프로그램 설명

이 프로그램을 실행시킬 때 DOS상태에서 그냥 'BASIC' 또는 'GWBASIC'을 입력하면 에러가 나기 쉽다. 72행에서 지정한 레코드의 길이 때문에 발생하는 에러인데 한 레코드의 길이가 527바이트나 되므로 일반 베이직에서는 'BASIC / S : 527'로 실행할 것을 권하며 만일 GWBASIC에서도 에러가 발생한다면 'GWBASIC / S : 527'로 실행시켜야 한다. 한글처리 문제도 여기서는 HP(DB).COM을 이용하고 있으나 그외 THP.COM, NKP.COM 등 독자들이 사용하는 베이직에서 실행가능한 한글을 사용하기 바란다.

처음 프로그램을 실행시키면 화면 1이 나온다. 리스트 1의 590번 문장과 830번 문장부터 실제의 초기 화면을 만들고 있으며 다음과 같은 방법으로 원하는 루틴을 입력받고 있다.

```
930 LOCATE 10,48
940 PRINT "원하는 번호는 : ";: OO$ = INPUT$(1)
950 OO = VAL (OO$)
960 ON OO GOTO 110,1860,1580,990:REM
    번호대로 분기함.
```

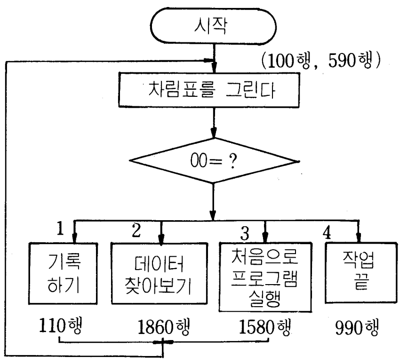
즉, 원하는 루틴번호는 INPUT$(1)에 의해 문자스트링형으로 입력되고 다시 VAL(OO$)을 거쳐 비로소 수치형 변수인 'OO'에 기억되고 있다. 그 후 960번 문장에서는 ON..GOTO문을 이용하고 있는데 선택한 번호가 1(레코드판 기록하기)이면 110번으로 분기하고 2(데이터 찾아보기)이면 1860번으로 분기하고... 이런식으로 원하는 루틴선택이 이루어지게 된다.

처음 프로그램 실행루틴

처음 실행루틴(선택번호 3번)은 보통 '화일의 초기화'라고도 하는 루틴인데, 앞으로 사용할 데이터화일을 미리 만들어 놓는 곳이다. 1580번부터 1850번 문장까지가 처음 프로그램 실행루틴인데 1580번에서 1700번 문장까지는 화일을 확실히 초기화시킬 것인지 재확인하는 곳이고, 1712번부터 실제로 빈 데이터 화일(내용이 없는 화일)을 만들고 있다.

행번호 70부터 화일을 열었으므로

〔그림 1〕 프로그램 구성



1730번의 FOR I=1 TO MAX에 의해 바로 한레코드씩 MAX만큼(200번)을 반복하여 빈 데이터화일을 만든다(그림 2 참조). 그리고 1790번에 PRINT CODE%명령이 있으므로 화면에 작성되고 있는 레코드번호가 표시된다.

참고로, 70번부터 시작되는 레코드 만들기에 대해서 알아 보면 먼저 OPEN문으로 527바이트를 할당한다. 그다음 FIELD문으로 입력번호 항목을 F1$으로 지정하여 3바이트를 할당(FIELD #1,3 AS F1$)하고 이름항목을 F2$로 지정하여 15바이트를 할당(,15 AS F2$)하고, 계속 이런 형태로 하여 총 527바이트 크기인 레코드를 만드는 원리이다(**보기 1** 참조).

기록하기 루틴

선택번호 1번에 해당하는 이 루틴은 말 그대로 코드와 입력번호, 이름, 레코드판 앞뒷면의 곡명 등을 입력받아 레코드(Record) 단위로 저장하는 작업을 수행한다(**보기 2** 참조).

이때 하나의 레코드를 이루는 항목(Item)들의 입력이 완료되면 '정확합니까'라는 확인메시지가 출력되는데 'Y'나 'y'를 누르면 데이터화일에 기록되고 그외의 다른 키를 누르면 화면을 전부 지우고 다음 레코드판의 입력을 준비한다(**그림 3** 참조).

IBM-PC 및 그 호환기종에서는 INPUT문으로 입력받은 내용을 화일버퍼로 옮길 때 이미 지정한 길이보다 긴 내용이 있으면 그 부분을 잘라버리므로 다음번 레코드에 겹쳐 써질 염려는 없다. 한 예로 코드번호는 80번 문장에서 9바이트로 할당되고 있으므로 'Korea-popsong' (13바이트)이라고 입력하면 화일버퍼에는 'Korea-pop'만 옮겨진다. 그리고 A면의 첫곡명은 25바이트로 할당되었으므로 '담다디'(6바이트)라고 입력되었다면 나머지 19바이트는 빈칸(Blank)으로 채워져서 화일버퍼로 옮겨지는 것이다. 한글은 한자가 2

〔화면 1〕 레코드판 관리 시작차림표 화면

```
                 ▓▓ 레코드판 관리 ▓▓

  1. 레코드판 관리 기록하기

  2. 데이타 찾아 보기

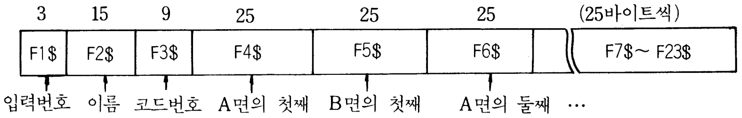
  3. 처음 프로그램을 실행시킬 때        원하는 번호는 : _

  4. 프로그램 끝내기
```

〔보기 1〕

```
1712  GOSUB 70 : REM open file
1720 PRINT TITLE$:REM 레코드관리
1730  FOR I=1 TO MAX:REM cf #60
1740    CODE% = I
1750  Z$="*":U$="*":A$="*":B$="*":C$="*":D$="*":E$="*":F$="*":G$="*":H$="*":I$="
*":J$="*":K$="*":L$="*":M$="*":N$="*":W$="*":P$="*":Q$="*":R$="*":S$="*":T$="*"
1760 LSET F0$=CHR$(255):LSET F1$=MKI$(CODE%):LSET F2$=Z$:LSET F3$=U$:LSET F4$=A$
:LSET F5$=B$:LSET F6$=C$:LSET F7$=D$:LSET F8$=E$:LSET F9$=F$:LSET F10$=G$
1770 LSET F11$=H$:LSET F12$=I$:LSET F13$=J$:LSET F14$=K$:LSET F15$=L$:LSET F16$=
M$:LSET F17$=N$:LSET F18$=W$:LSET F19$=P$:LSET F20$=Q$:LSET F21$=R$:LSET F22$=S$
:LSET F23$=T$
1780 PUT #1,CODE%
1790 PRINT CODE%,
1800 NEXT I:PRINT
```

〔그림 2〕 데이터 화일(LKW-TEL)의 구조



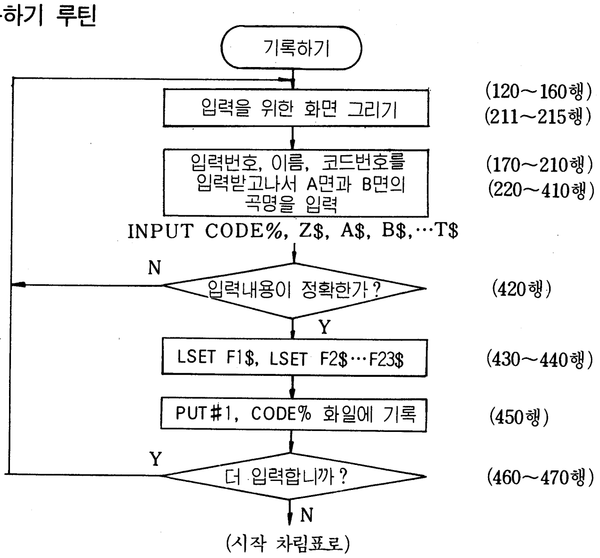
LKW-TEL화일의 총 길이는 527바이트임 (3+15+9+25*20=527)

〔보기 2〕

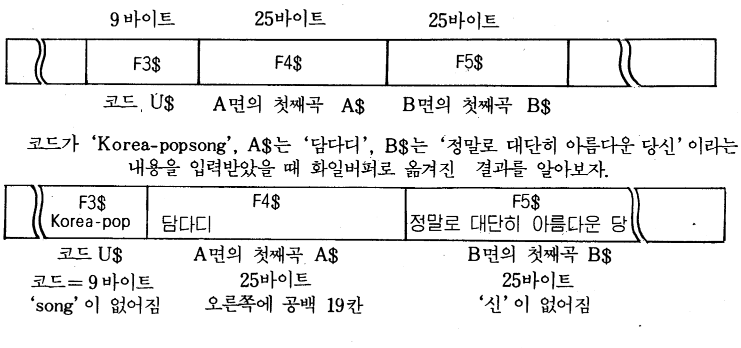
```
102 REM ***** input data *****
110 CLS:GOSUB 70 : REM open file
170 LOCATE 8,20:PRINT N1$;:INPUT":",CODE% :REM 입력 번호
180 IF CODE%=0 THEN CLS:GOTO 460
190 LOCATE 2,65:PRINT"          "
200 LOCATE 10,20:PRINT N2$;:INPUT ":",Z$ :REM      이름
210 LOCATE 12,20:PRINT N3$;:INPUT ":",U$:CLS: REM 코드 번호
```

```
                          :
214 LOCATE 3,15:COLOR 0,7:PRINT"▓A 면 ▓":COLOR 7,0
215 LOCATE 3,50:COLOR 0,7:PRINT"▓B 면 ▓":COLOR 7,0
220 LOCATE 5,6:PRINT N4$;:INPUT ":",A$
230 LOCATE 5,41:PRINT N5$;:INPUT ":",B$
240 LOCATE 6,6:PRINT N6$;:INPUT ":",C$
                          :
390 LOCATE 13,41:PRINT N21$;:INPUT ":",R$
400 LOCATE 14,6:PRINT N22$;:INPUT ":",S$
410 LOCATE 14,41:PRINT N23$;:INPUT ":",T$
420 LOCATE 18,5:INPUT"정확합니까 [Y/N]:",JUNG$:IF JUNG$="Y" OR JUNG$="y" THEN 43
0 ELSE CLS:GOTO 120
430 LSET F0$=CHR$(0):LSET F1$=MKI$(CODE%):LSET F2$=Z$:LSET F3$=U$:LSET F4$=A$:LS
ET F5$=B$:LSET F6$=C$:LSET F7$=D$:LSET F8$=E$:LSET F9$=F$:LSET F10$=G$
440 LSET F11$=H$:LSET F12$=I$:LSET F13$=J$:LSET F14$=K$:LSET F15$=L$:LSET F16$=M
$:LSET F17$=N$:LSET F18$=W$:LSET F19$=P$:LSET F20$=Q$:LSET F21$=R$:LSET F22$=S$:
LSET F23$=T$
450 PUT #1,CODE%
460 LOCATE 20,5:PRINT "더하십니까? [Y/N]:";:LK$=INPUT$(1):PRINT LK$
```

〔그림 3〕 기록하기 루틴



〔그림 4〕 화일버퍼로 옮기기



바이트이고 띄어쓰기는 1바이트, 숫자나 알파벳은 한자가 1바이트를 차지하는 것에 유의하면 바이트 수를 계산하기가 쉬울 것이다(그림 4 참조).

데이터 찾아 보기 루틴

시작 차림표에서 데이터 찾아보기(2번)를 선택하면 컴퓨터는 1860행의 찾아보기 루틴을 실행시킨다(화면 2 참조). 이 루틴에서는 다시 'CC'변수를 이용해서 레코드로 접근하는 방법을 선택하게 하였다.

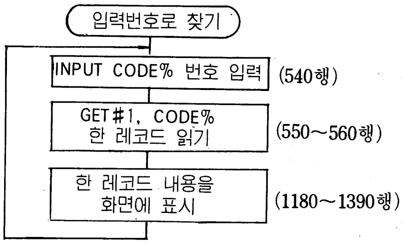
```
ON CC GOTO 500,1030,2120,2230,710,710
```

데이터를 찾는 여러가지 방법중에서 특히 입력번호로 찾는 루틴은 랜덤화일에서 데이터를 찾는 표준적인 방법이므로 잘 알아 두기 바란다.

```
530 REM
540 PRINT N1$;:INPUT " : ",CODE%:REM 입력번호
550 GET #1, CODE%
560 O%=CVI(F1$):Z$=F2$:U$=F3$:A$=F4$:B$
    =F5$:C$=F6$:D$=F7$:E$=F8$:F$=F9$:G$=
     F10$: H$=F11$:I$=F12$:J$=F13$:K$=F14$
     :L$=F15$:M$=F16$:N$=F17$:W$=F18$:P$=
     F19$:Q$=F20$:R$=F21$:S$=F22$:T$=F23$
570 GOSUB 1180
```

즉 GET #1, CODE%에 의해 지정된 코드번호(CODE%)에 해당하는 레코드를 찾아(GET #1) 1180번 문장으로 분기한 후 그 내용(560번 문장)을 화면에 보여주는 원리이다(그림 5와 보기 3 참조).

〔그림 5〕 데이터 찾기 루틴 흐름도



〔화면 2〕 데이터 찾기 화면

```
            ▓ 데이타 찾기 ▓

▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
▓                                  ▓
▓ [1]. 입력 번호로 찾기              ▓
▓                                  ▓
▓ [2]. 이름으로 찾기                 ▓
▓                                  ▓
▓ [3]. 전화 번호로 찾기              ▓
▓                                  ▓
▓ [4]. 연속하여 찾아보기             ▓
▓                                  ▓
▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓

   6 = 시작 차림표로    ............ 원하는 번호는 : _
```

〔보기 3〕

```
1180 CLS:REM *** 화면에 찾은 데이타 보여주기 **********
1190 O$=STR$(O%): LOCATE 2,20:PRINT N1$;" : ";O$:REM O$=입력 번호
1191 LOCATE 3,20:PRINT N2$;" : "   :LOCATE 3,35:PRINT Z$ :REM Z$=이름
1192 LOCATE 4,20:PRINT N3$;" : "   :LOCATE 4,35:PRINT U$:REM   U$=코드번호
1193  LOCATE 7,3:PRINT"▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓▓
   ▓▓▓▓▓▓▓▓▓▓"
1194 FOR K=7 TO 18:LOCATE K,3:PRINT"▓":LOCATE K,38:PRINT"▓":LOCATE K,77
    :PRINT" ▓":NEXT
1196 FOR K= 76 TO 3 STEP-1:LOCATE 18,K:PRINT"▓":NEXT K
1230 REM
1240 REM
1242 REM
1250 LOCATE 8,6:PRINT N4$;" : ";A$:LOCATE 8,41:PRINT N5$;" : ";B$:REM
    A$= A면의 첫째 곡명
1260 LOCATE 9,6:PRINT N6$;" : ";C$:LOCATE 9,41:PRINT N7$;" : ";D$
1270 LOCATE 10,6:PRINT N8$;" : ";E$:LOCATE 10,41:PRINT N9$;" : ";F$
1280 LOCATE 11,6:PRINT N10$;" : ";G$:LOCATE 11,41:PRINT N11$;" : ";H$
1290 LOCATE 12,6:PRINT N12$;" : ";I$:LOCATE 12,41:PRINT N13$;" : ";J$
1300 LOCATE 13,6:PRINT N14$;" : ";K$:LOCATE 13,41:PRINT N15$;" : ";L$
1310 LOCATE 14,6:PRINT N16$;" : ";M$:LOCATE 14,41:PRINT N17$;" : ";N$
1320 LOCATE 15,6:PRINT N18$;" : ";W$:LOCATE 15,41:PRINT N19$;" : ";P$
1330 LOCATE 16,6:PRINT N20$;" : ";Q$:LOCATE 16,41:PRINT N21$;" : ";R$
1340 LOCATE 17,6:PRINT N22$;" : ";S$:LOCATE 17,41:PRINT N23$;" : ";T$
1350 LOCATE 22:INPUT"프린터로 찍으시겠읍니까?[Y/N]:",PR$
1360 IF PR$ ="Y" OR PR$="y" THEN GOSUB 1410
1370 LOCATE 22,40:INPUT"또있읍니까?[Y/N](다시 찾기 = A):",LY$
1372 IF LY$= "Y" OR  LY$=  "y" THEN  RETURN : REM 또 있음.
1380 IF LY$<>"Y" THEN IF LY$<> "y" THEN GOTO 710: REM 710행은 시작차림표임.
1390 RETURN
```

〔리스트 1〕 레코드판 관리 프로그램 리스트

```
6 REM basic/s:527로 시작해야 함.
7 REM  must start basic/s:527 if not error
8 REM original pg by Leeks.
9 REM updated by    Jo Namhoon.
10 REM save"r"
12 N1$="입력 번호":N2$="이 름":N3$="코드 번호"
20 N4$="[ 1 ]":N5$="[ 1 ]":N6$="[ 2 ]":N7$="[ 2 ]":N8$="[ 3 ]":
   N9$="[ 3 ]":N10$="[ 4 ]":N11$="[ 4 ]":N12$="[ 5 ]": N13$="[
   5 ]": N14$="[ 6 ]": N15$="[ 6 ]": N16$="[ 7 ]":N17$="[ 7 ]":
   N18$="[ 8 ]"
30 N23$="[ A ]"
40 TITLE$="레코드판 관리":CLS:KEY OFF
50 FILE$="lkw-tel"
60 MAX=200
62 GOTO 100 :REM 차림표로 감.
70 CLOSE
72 OPEN "R",#1,FILE$,527
80 FIELD #1,3 AS F1$,15 AS F2$,9 AS F3$,25 AS F4$,25 AS F5$,25
   AS F6$,25 AS F7$, 25 AS F8$,25 AS F9$,25 AS F10$,25 AS F11$
90 FIELD #1,227 AS DUMMY$, 25 AS F12$, 25 AS F13$, 25 AS F14$,
   25 AS F15$,25 AS F16$,25 AS F17$,25 AS F18$,25  AS  F19$,25
   AS F20$,25 AS F21$,25 AS F22$,25 AS F23$
91 RETURN :REM
92 REM FIELD #1,200 AS CHUEUM$, 77 AS DAEUM$,  25 AS F14$,25 AS
   F15$,25 AS F16$,25 AS F17$,25 AS F18$,25 AS F19$,25 AS F20$,
   25 AS F21$,25 AS F22$,25 AS F23$
100 GOTO 590 :REM 시작 차림표로.
102 REM ***** input data *****
110 CLS:GOSUB 70 : REM open file
120 FOR I=16 TO 55:LOCATE 6,I:PRINT"▓":NEXT I
130 FOR I=7 TO 14:LOCATE I,16:PRINT"▓":LOCATE I,55:PRINT"▓":
    NEXT I
140 FOR I=16 TO 55:LOCATE 14,I:PRINT"▓":NEXT I
150 LOCATE 8,20:PRINT N1$:LOCATE 10,20:PRINT N2$:LOCATE 12,20:
    PRINT N3$
160 LOCATE 2,65:PRINT "0=입력 끝"
170 LOCATE 8,20:PRINT N1$;:INPUT":",CODE% :REM 입력 번호
180 IF CODE%=0 THEN CLS:GOTO 460
```

```
190 LOCATE 2,65:PRINT"
200 LOCATE 10,20:PRINT N2$;:INPUT ":",Z$ :REM      이름
210 LOCATE 12,20:PRINT N3$;:INPUT ":",U$:CLS: REM 코드 번호
211 FOR I=3 TO 76:LOCATE 3,I:PRINT"▒":NEXT I
212 FOR I=3 TO 16 :LOCATE I,3:PRINT"▒":LOCATE I,38:PRINT"▒":
    LOCATE I,76:PRINT"▒":NEXT I
213 FOR I=76 TO 3 STEP-1:LOCATE 16,I:PRINT"▒":NEXT I
214 LOCATE 3,15:COLOR 0,7:PRINT"▒A 면 ▒":COLOR 7,0
215 LOCATE 3,50:COLOR 0,7:PRINT"▒B 면 ▒":COLOR 7,0
220 LOCATE 5,6:PRINT N4$;:INPUT ":",A$
230 LOCATE 5,41:PRINT N5$;:INPUT ":",B$
240 LOCATE 6,6:PRINT N6$;:INPUT ":",C$
250 LOCATE 6,41:PRINT N7$;:INPUT ":",D$
260 LOCATE 7,6:PRINT N8$;:INPUT ":",E$
270 LOCATE 7,41:PRINT N9$;:INPUT ":",F$
280 LOCATE 8,6:PRINT N10$;:INPUT ":",G$
290 LOCATE 8,41:PRINT N11$;:INPUT ":",H$
300 LOCATE 9,6:PRINT N12$;:INPUT ":",I$
310 LOCATE 9,41:PRINT N13$;:INPUT ":",J$
320 LOCATE 10,6:PRINT N14$;:INPUT ":",K$
330 LOCATE 10,41:PRINT N15$;:INPUT ":",L$
340 LOCATE 11,6:PRINT N16$;:INPUT ":",M$
350 LOCATE 11,41:PRINT N17$;:INPUT ":",N$
360 LOCATE 12,6:PRINT N18$;:INPUT ":",W$
370 LOCATE 12,41:PRINT N19$;:INPUT ":",P$
380 LOCATE 13,6:PRINT N20$;:INPUT ":",Q$
390 LOCATE 13,41:PRINT N21$;:INPUT ":",R$
400 LOCATE 14,6:PRINT N22$;:INPUT ":",S$
410 LOCATE 14,41:PRINT N23$;:INPUT ":",T$
420 LOCATE 18,5:INPUT"정확합니까 [Y/N]:",JUNG$:IF JUNG$="Y" OR
    JUNG$="y" THEN 430 ELSE CLS:GOTO 120
430 LSET F0$=CHR$(0):LSET F1$=MKI$(CODE%):LSET F2$=Z$:LSET F3$=
    U$:LSET F4$=A$:LSET F5$=B$:LSET F6$=C$:LSET F7$=D$:LSET F8$
    =E$:LSET F9$=F$:LSET F10$=G$
440 LSET F11$=H$:LSET F12$=I$:LSET F13$=J$:LSET F14$ = K$: LSET
    F15$=L$:LSET F16$=M$:LSET F17$=N$:LSET F18$=W$:LSET F19$=P$:
    LSET F20$=Q$:LSET F21$=R$:LSET F22$=S$:LSET F23$=T$
450 PUT #1,CODE%
460 LOCATE 20,5:PRINT "더하십니까? [Y/N]:";:LK$=INPUT$(1):PRINT LK$
470 IF LK$ = "y" OR LK$ = "Y" THEN 480 ELSE GOTO 710
480 CLS:GOTO 120
490 REM 입력 번호로 찾는 프로그램
500 CLS
510 REM
520 CLS:REM 화면지우기
530 REM
540 PRINT N1$;:INPUT " : ",CODE%:REM 입력 번호
550 GET #1, CODE%
560 O%=CVI(F1$):Z$=F2$:U$=F3$:A$=F4$:B$=F5$:C$=F6$:D$=F7$:E$=
    F8$:F$=F9$:G$=F10$:H$=F11$:I$=F12$:J$=F13$:K$=F14$:L$=F15$
    :M$=F16$:N$=F17$:W$=F18$:P$=F19$:Q$=F20$:R$=F21$:S$=F22$:T$
    =F23$
570 GOSUB 1180
580 GOTO 520
590 REM 시작 차림표 루틴 *************************************
600 CLS
700 REM
710 CLS
720 LOCATE 2,20
730 COLOR 0,7:PRINT "▒▒";TITLE$;"▒▒":COLOR 7,0: REM 레코드판 관리
740 REM
750 GOTO 830
760 LOCATE 6,3
770 LOCATE 4,3:PRINT"████████████████████████████████████████
    ███████████████"
780 FOR I=5 TO 15
790     LOCATE I,3 : PRINT "▒":LOCATE I,43:PRINT "▒":LOCATE
        I,71:PRINT"▒"
800 NEXT I
810  RETURN
820 REM ******************************************************
830 LOCATE 6,7
840 PRINT "[1]. ";"레코드판 기록하기":REM #110
850 LOCATE 8,7
860 PRINT "[2]. 데이타 찾아보기": REM #1860
870 LOCATE 10,7
880 PRINT "[3]. 처음 프로그램을 실행 시킬때":REM #1580
890 LOCATE 12,7
900 PRINT "[4]. 프로그램 끝내기": REM #990
910 GOSUB 760: GOTO 920:REM ************************************
920 REM  FOR I=3 TO 70:LOCATE 15,I:PRINT"▒":NEXT I
922 LOCATE 15,3: PRINT"████████████████████████████████████████
    ████████████████ "
930 LOCATE 10,48
940 PRINT "원하는 번호는 : ";:  OO$ = INPUT$(1)
950 OO = VAL (OO$)
960 ON OO GOTO 110,1860,1580,990:REM 번호대로 분기함.
970 PRINT CHR$(7):GOTO 940
980 REM
990 REM 마무리
991 CLS
1000 LOCATE 10,25:COLOR 0,7:PRINT"▒                    ▒":
     COLOR 7,0
1010           LOCATE 11,25:PRINT"▒ 감사합니다... ▒"
1011 LOCATE 12,25:COLOR 0,7:PRINT"▒                    ▒":
     COLOR 7,0
1012 CLOSE
1020 END
1030 REM 이름으로 찾기
1040 REM
1050 CLS
1060 PRINT N2$;" 첫자만 알때='+', 아니면 '리턴'을.";:INPUT " : ",CHUT$
1070 IF CHUT$ = "+" THEN PRINT N2$;" 첫자는 ";:INPUT" : ",IC$:
     GOTO 1090
```

```
1080 PRINT "입력시킨";N2$;: INPUT " : ",IN$
1090 FOR I = 1 TO MAX:REM cf #7:
1100       GET #1,I
1110       O%=CVI(F1$): Z$=F2$: U$=F3$:A$=F4$:B$=F5$:C$=F6$:D$=
           F7$:E$=F8$:F$=F9$:G$=F10$:H$=F11$:I$=F12$:J$=F13$:K$
           =F14$:L$=F15$:M$=F16$:N$=F17$:W$=F18$:P$=F19$:Q$=F20$
           :R$=
1120       LOCATE 1,76:PRINT I
1130     IF IN$ = LEFT$(Z$,LEN(IN$)) AND CHUT$ <> "+" THEN
         GOSUB 2310:GOTO 1160
1140     NC$ = Z$:REM 첫자용
1150     NC$ = LEFT$(NC$, LEN(IC$)):IF CHUT$="+" AND NC$=IC$
         THEN GOSUB 1180:REM 첫자가 같은것 만났음
1160 NEXT I
1170 GOTO 710
1180 CLS:REM *** 화면에 찾은 데이타 보여주기 **********
1190 O$=STR$(O%): LOCATE 2,20:PRINT N1$;" : ";O$:REM O$=입력 번호
1191 LOCATE 3,20:PRINT N2$;" : "  :LOCATE 3,35:PRINT Z$ :REM
     Z$=이름
1192 LOCATE 4,20:PRINT N3$;" : "  :LOCATE 4,35:PRINT U$:REM
     U$=코드번호
1193  LOCATE 7,3:PRINT"████████████████████████████████████
█████████████████"
1194 FOR K=7 TO 18:LOCATE K,3:PRINT"█":LOCATE K,38:PRINT"█":
     LOCATE K,77:PRINT"█":NEXT
1196 FOR K= 76 TO 3 STEP-1:LOCATE 18,K:PRINT"█":NEXT K
1230 REM
1240 REM
1242 REM
1250 LOCATE 8,6:PRINT N4$;" : ";A$:LOCATE 8,41:PRINT N5$;" : ";
     B$:REM A$= A면의  첫째 곡명
1260 LOCATE 9,6:PRINT N6$;" : ";C$:LOCATE 9,41:PRINT N7$;" : ";
     D$
1270 LOCATE 10,6:PRINT N8$;" : ";E$:LOCATE 10,41:PRINT N9$;" :
     ";F$
1280 LOCATE 11,6:PRINT N10$;" : ";G$:LOCATE 11,41:PRINT N11$;"
      : ";H$
1290 LOCATE 12,6:PRINT N12$;" : ";I$:LOCATE 12,41:PRINT N13$;"
      : ";J$
1300 LOCATE 13,6:PRINT N14$;" : ";K$:LOCATE 13,41:PRINT N15$;"
      : ";L$
1310 LOCATE 14,6:PRINT N16$;" : ";M$:LOCATE 14,41:PRINT N17$;"
      : ";N$
1320 LOCATE 15,6:PRINT N18$;" : ";W$:LOCATE 15,41:PRINT N19$;"
      : ";P$
1330 LOCATE 16,6:PRINT N20$;" : ";Q$:LOCATE 16,41:PRINT N21$;"
      : ";R$
1340 LOCATE 17,6:PRINT N22$;" : ";S$:LOCATE 17,41:PRINT N23$;"
      : ";T$
1350 LOCATE 22:INPUT"프린터로 찍으시겠습니까?[Y/N]:",PR$
1360 IF PR$ ="Y" OR PR$="y" THEN GOSUB 1410
1370 LOCATE 22,40:INPUT"또있읍니까?[Y/N](다시 찾기 = A):",LY$
1372 IF LY$= "Y" OR  LY$=  "y" THEN  RETURN : REM 또 있음.
1380 IF LY$<>"Y" THEN IF LY$<> "y" THEN GOTO 710: REM 710행은
     시작자립표임.
1390 RETURN
1400 GOTO 1050
1410 REM 프린터로 찍기
1411 PRINT "프린터를 켜세요..."
1412 LPRINT CHR$(27)+"h"+CHR$(1),
1414 LPRINT CHR$(27)+"g"+CHR$(1)
1420 LPRINT CHR$(10)
1430 LPRINT "--------------------------------------------------
--------"
1440 LPRINT N1$;TAB(20);O$:LPRINT N2$;TAB(20);Z$:LPRINT N3$;
     TAB(20);U$
1450 LPRINT N4$;TAB(10);A$;TAB(40);N5$;TAB(50);B$
1460 LPRINT N6$;TAB(10);C$;TAB(40);N7$;TAB(50);D$
1470 LPRINT N8$;TAB(10);E$;TAB(40);N9$;TAB(50);F$
1480 LPRINT N10$;TAB(10);G$;TAB(40);N11$;TAB(50);H$
1490 LPRINT N12$;TAB(10);I$;TAB(40); N13$;TAB(50);J$
1500 LPRINT N14$;TAB(10);K$;TAB(40);N15$;TAB(50);L$
1510 LPRINT N16$;TAB(10);M$;TAB(40); N17$;TAB(50);N$
1520 LPRINT N18$;TAB(10);W$;TAB(40);N19$;TAB(50);P$
1530 LPRINT N20$;TAB(10);Q$;TAB(40);N21$;TAB(50);R$
1540 LPRINT N22$;TAB(10);S$;TAB(40);N23$;TAB(50);T$
1550 LPRINT "--------------------------------------------------
--------"
1560 REM 다시 화면으로
1570 RETURN
1580 REM 처음 프로그램을 실행 할때
1590 CLS:PRINT CHR$(7):REM 벨
1600 PRINT
1610 PRINT "   다시한번 확인 하세요. "
1620 PRINT
1630 PRINT "   잘못하면 모든 데이타를 잃어버립니다."
1640 PRINT
1650 PRINT "   빈 화일 만들 이름은 ";FILE$;"입니다."
1660 PRINT
1670 PRINT : PRINT
1680 INPUT " 정말로 빈화일을 만듭니까? [Y/N]:",QW$
1690 IF QW$ ="Y" OR QW$="y" THEN GOTO 1710
1700 GOTO 710
1710 CLS
1712  GOSUB 70 : REM open file
1720 PRINT TITLE$:REM 레코드관리
1730  FOR I=1 TO MAX:REM cf #60
1740    CODE% = I
1750  Z$="*":U$="*":A$="*":B$="*":C$="*":D$="*":E$="*":F$="*":
      G$="*":H$="*":I$="*":J$="*":K$="*":L$="*":M$="*":N$="*":
      W$="*":P$="*":Q$="*":R$="*":S$="*":T$="*"
1760 LSET F0$=CHR$(255):LSET F1$=MKI$(CODE%):LSET F2$=Z$:LSET
     F3$=U$:LSET F4$=A$:LSET F5$=B$:LSET F6$=C$: LSET F7$=D$:
```

```
     LSET F8$=E$:LSET F9$=F$:LSET F10$=G$
1770 LSET F11$=H$:LSET F12$=I$:LSET F13$=J$:LSET F14$=K$:LSET
     F15$=L$:LSET F16$=M$:LSET F17$=N$:LSET F18$=W$:LSET F19$
     =P$:LSET F20$=Q$:LSET F21$=R$:LSET F22$=S$:LSET F23$=T$
1780 PUT #1,CODE%
1790 PRINT CODE%,
1800 NEXT I:PRINT
1810 PRINT
1820 PRINT "빈 화일을 다 만들었습니다..."
1830 FOR Z = 1 TO 1000
1840 NEXT Z
1850 GOTO 710
1860 REM 데이타 찾아보기 메뉴판
1870 CLS : GOSUB 70 : REM  open file
1880 LOCATE 2,11
1890 COLOR 0,7:PRINT "▒▒▒ * 데이타 찾기 * ▒▒▒":COLOR 7,0
1910 LOCATE 4,3:PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
▒▒▒▒▒▒▒▒"
1920 FOR I=5 TO 15:LOCATE I,3:PRINT"▒":LOCATE I,43:PRINT"▒":
     LOCATE I,71:PRINT"▒":NEXT I
1921 LOCATE 15,3 :PRINT"▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒
▒▒▒▒▒▒▒▒"
1922 REM FOR I=3 TO 69:LOCATE 15,I:PRINT"▒":NEXT I
1940 LOCATE 6,7
1950 PRINT"[1]. ";N1$;" 로 찾기":REM 입력 번호 #500
1960 LOCATE 8,7
1970 PRINT"[2]. ";"이름으로 찾기"
1980 LOCATE 10,7
1990 PRINT"[3]. ";"코드 번호로 찾기"
2000 LOCATE 12,7
2010 PRINT"[4]. 연속하여 찾아보기"
2020 REM
2030 REM
2040 GOSUB 780
2070 LOCATE 17,7
2080 PRINT "6= 시작 차림표로  ..................원하는 번호는:";:
     CC$=INPUT$(1)
2090 CC = VAL(CC$)
2100 ON CC GOTO 500,1030,2120,2230,710,710
2110 PRINT CHR$(7):GOTO 2070
2120 REM 코드 번호로 찾기
2130 CLS
2140 REM
2150 PRINT "입력시킨";N3$;:INPUT ":",IT$
2160 FOR KS=1 TO MAX
2170    GET #1,KS
2180    O%=CVI(F1$):Z$=F2$:U$=F3$:A$=F4$:B$=F5$:C$=F6$:D$=F7$:
        E$=F8$:F$=F9$:G$=F10$:H$=F11$:I$=F12$:J$=F13$:K$=F14$:
        L$=F15$:M$=F16$:N$=F17$:W$=F18$:P$=F19$:Q$=F20$:R$=F21$
        :S$=F22
2190    LOCATE 1,76:PRINT KS
2200    IF IT$= LEFT$(U$,LEN(IT$)) THEN GOSUB 1180:REM
2210 NEXT KS
2220 GOTO 710
2230 REM 연속하여보여주기
2240 L = 1
2250 IF L > MAX THEN 2430
2260    GET #1,L
2270    O%=CVI(F1$):Z$=F2$:U$=F3$:A$=F4$:B$=F5$:C$=F6$:D$=F7$:
        E$=F8$:F$=F9$:G$=F10$:H$=F11$:I$=F12$:J$=F13$:K$=F14$:
        L$=F15$:M$=F16$:N$=F17$:W$=F18$:P$=F19$:Q$=F20$:R$=F21$
        :S$=F22
2280    LOCATE 1,76:PRINT L
2290 GOSUB 1180
2300 L = L + 1:GOTO 2250
2310 REM 루틴 명령어
2320 GOSUB 1180
2330 IF LY$ = "a" OR LY$ ="A" THEN 1400
2340 RETURN : REM to #3052
2350 REM 에러가 났을 때
2360 PRINT CHR$(7)
2370 PRINT
2380 STOP
2390 ERROR ERR
2400 FOR I = 1 TO 1500:NEXT I
2410 REM
2420 GOTO 710
2430 REM 10개를다 찾았을 때
2440 GOSUB 2450:GOTO 2500
2450 REM
2460 PRINT:LOCATE 22,1:PRINT SPACE$(70):LOCATE 23,1:PRINT
     SPACE$(70)
2470 LOCATE 22,5:COLOR 0,7:REM 인버스 글자로
2480 PRINT "계속 하시겠읍니까 [Y/N] ";:LK$ = INPUT$(1):COLOR 7,
     0:PRINT LK$
2490 RETURN
2500 IF LK$="Y" OR LK$="y" THEN 2520
2510 GOTO 710
2520 REM 데이타를 화면에 나타내기
2750 REM
3000 PRINT"프린터를 켜세요":BEEP:PRINT"프린터 종류는 엡슨(LQ-1500) =
     1,큐닉스(PC-24)=0 "
3010 QNIX$=INPUT$(1):IF QNIX$="0" THEN LPRINT CHR$(27);"h";
     CHR$(1)    ELSE        LPRINT CHR$(27);"h"
3260 REM :eof:hk9-tel8.bas
3300 REM :EOF:reco-9.bas
```

명령어의 인수와 PSP

이용일 /한국전자통신연구소

MS-DOS상에서 프로그램을 개발하는 경우에는, 프로그램이 DOS로부터 어떤 정보와 함께 기동되며, 프로그램이 받는 서비스의 내용이 무엇인가를 알아야 한다. C언어에 의한 개발을 하는 경우, 대부분의 핵심내용은 라이브러리 및 초기화루틴에 숨어 있기 때문에, 그 자세한 내용을 모르더라도 어느 정도의 프로그램은 작성할 수 있지만, 시스템(DOS)에 의존하는 프로그램을 작성하는 경우에는, 직접 펑션콜을 이용할 필요가 있기 때문에, DOS의 지식없이는 사용하기 쉬운 프로그램을 작성할 수 없다. 또, 하드웨어를 직접 제어하는 프로그램을 작성하는 경우에, MS-DOS의 시스템과 프로그램의 연계에 대해 잘 알지 못하면, 하드웨어 시스템에 의존하는 이식성이 매우 낮은 프로그램이 되어 버린다.

본회에서는, C언어 및 어셈블리언어의 프로그램을 기술할 때 필요한, 프로그램과 DOS사이에 준비되어 있는 연계성에 대해 설명할 것이다. 또, DOS에 밀착한 프로그램에서는 빠질 수 없는 어셈블리언어의 프로그램과 C언어 프로그램과의 링크방법에 대해서도 설명할 것이다.

MS-DOS와 프로그램 인터페이스의 하나로서, 명령어라인의 인수를 프로그램에 넘기는 기능을 들 수 있다. 또 C언어로 프로그램을 개발하고 있는 경우에는 C의 초기화루틴에 의해 명령어라인이 어떻게 처리되어 프로그램에 넘어가는 가를 이해하는 것도 중요한 문제이다. 그래서 MS-DOS의 인수를 넘기는 방법과 함께 C의 초기화루틴 처리에 대해 설명하고, 또 그들을 이용하여 어떻게 명령어라인을 해석하는 가를 설명하겠다.

PSP

MS-DOS가 프로세스(메모리 중에 로드되어 실행가능한 프로그램)를 기동할 때는, 프로세스에 명령어라인의 인수를 넘기는 영역이 필요하다. 이 영역을 MS-DOS에서는 PSP(Program Segment Prefix)라 부르고, 프로세스가 로드된 메모리의 선두 256바이트에 놓여진다. PSP에는 명령어라인의 인수 뿐만 아니라, 프로세스의 처리에 필요한 정보가 들어있기 때문에, 이 영역은 프로세스 기동시에, DOS와 프로세스의 인터페이스영역으로서의 역할을 한다.

PSP에 격납되어 있는 정보는, 기동시에 프로로세스에 전해지는 것만은 아니다. MS-DOS는 한번에 하나의 프로세스 밖에 실행할 수 없는 싱글태스크(Single Task)인 DOS이지만, 임의의 한 프로세스에서 별도의 다른 프로세스를 기동할 수 있어, 결국 한번에 복수의 프로세스가 존재하는 경우가 있으므로 각 프로세스를 관리하기 위한 정보를 DOS의 내부에 갖고 있으면 예기치 못한 상황이 발생한다. 왜냐하면 그것은 프로세스의 최대 갯수를 알지못하기 때문에 적절한 수와 양의 관리영역을 미리 확보할 수 없기 때문이다. MS-DOS에서는 이 문제를 해결하기 위해 PSP의 내부에 프로세스의 관리정보를 기억시키고 있다. PSP를 한 프로세스에 하나씩 고유하게 갖고 있게 하면 위에서 열거한 문제는 발생하지 않기 때문이다.

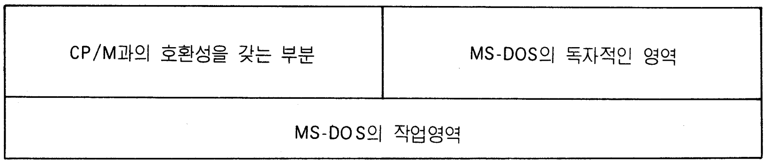
결국 PSP는 2가지 이유로 존재하는 것이 된다. 하나는, 기동된 프로세스에 대하여 인수 등 프로세스의 각종 실행환경을 넘기기 위해 준비되어 있고, 다른 한가지 이유는 MS-DOS가 그 프로세스를 관리하는 정보를 갖기 위해 이용된다는 것이다.

PSP에 들어있는 프로세스의 실행환경은, 2종류로 나누어 생각할 수 있다. 하나는 CP/M과의 호환성을 갖기위해 준비되어 있는 정보이고, 다른 하나는 MS-DOS 독자적인 실행환경의 정보이다. 그림 1에 PSP의 개략적인 내용을 보인다.

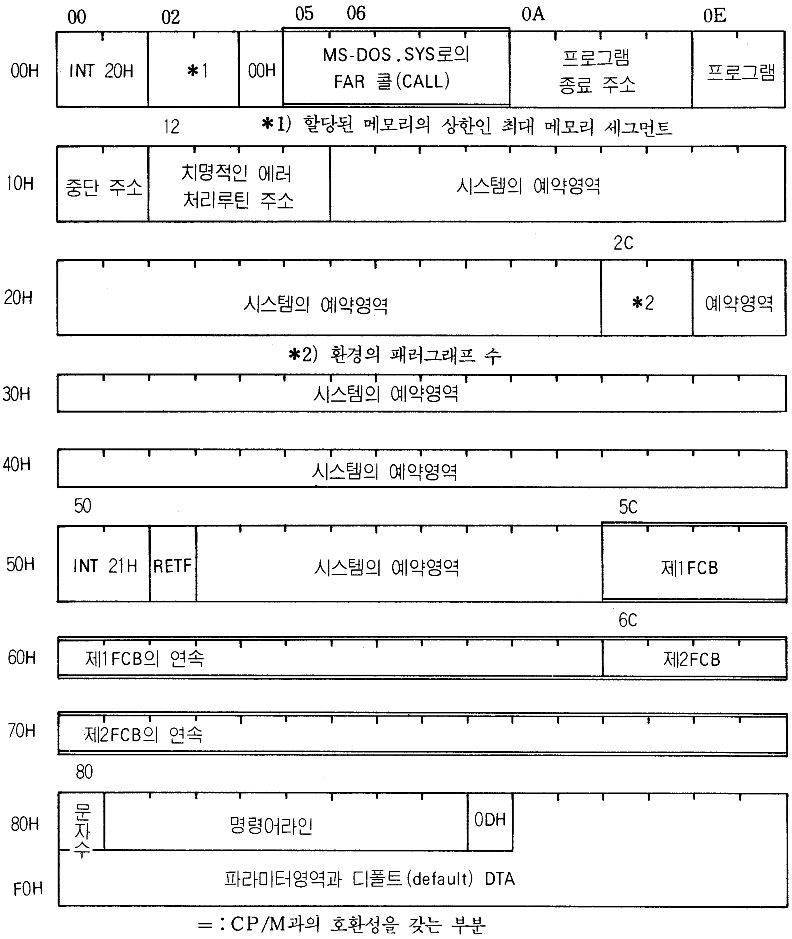
MS-DOS의 처음 버전은, 그 당시 널리 사용되고 있었던 CP/M상의 프로그램을 쉽게 이식할 수 있도록 하는 것을 목표로 개발되었던 DOS이었기 때문에, PSP의 내용도 CP/M의 스크래치 패드 영역(Scratch Pad Area)과의 호환성을 갖도록 그리고 확장부분에 있어서도 CP/M과 같은 개념의 토대위에 설계되었다. 그런데, MS-DOS가 버전 2.0으로 확장하면서 "환경"이라는 개념 및 계층화 디렉토리가 추가되었기 때문에, PSP의 양식(Format)도 이전의 MS-DOS(버전1.25 이전)와의 호환성을 가지면서 확장되었고, 이전의 PSP에서는 사용하지 않았던 영역에, 확장에 필요한 정보를 넣었다.

그림 2가 PSP의 세부내용을 나타낸 것인데, 05H-09H 및 5CH-7FH 사이의 영역이 CP/M과의 호환성을 가진 부분이고, 그 이외의 부분이 버전 2.0 이후의 MS-DOS에서 의미가 있는 영역이다. 그림 3은 프로그램 2의 PSP실예를 보인 것이고, 여기서 typen.exe는 주어진 텍스트화일에 문번호를 붙이는 실행형 화일이다. 프로그램에 대한 자세한 내용은 뒤에 설명한다.

〔그림 1〕 PSP의 내용



〔그림 2〕 PSP의 구체적인 구조



인수의 교환

PSP의 정보를 직접 액세스할 필요가 있는 것은 주로 어셈블리 언어에 의해 프로그램을 개발하는 경우이고, C 언어에 의해 프로그램을 개발하는 경우 PSP에 들어있는 정보의 대부분은, C의 초기화루틴에 의해 처리되기 때문에, 라이브러리 함수를 이용하는 것에 의해 그 존재를 망각할 수도 있다. 또 명령어의 인수도 초기화루틴에 의해 처리를 한후 프로그램에 넘겨지기 때문에, 프로그램에서 하는 처리는 일단 분해된 인수를 해석하는 것으로 충분하지만, 그때 초기화루틴이 어떤 규칙에 따라 명령어라인을 분해하는가를 파악해 두지 않으면 안된다.

초기화루틴에 의해 인수를 처리하는 것은 처리계에 따라 다르지만, 마이크로소프트 C의 경우 다음과 같은 규칙에 따라 분해된다.

① 각 인수는 공백문자(스페이스와 텝 문자)에 의해 구분된다.
②각 인수중에 와일드문자가 있는 경우에는 전개된다. (와일드문자 전개루틴(SETRAG. OBJ)을 링크한 경우에 만)
③ 인수를 ""로 구분하는 것에 의해 와일드문자 및 공백문자를 그대로 인수로서 넘길 수 있다.
④ ""문자 직전에 백슬래시(\)를 두는 것에 의해 그 기능을 취소할 수 있다.

여기에서 명령어라인이 어떻게 분해되는 가를 확인하기 위해 프로그램을 작성하여 실행해 보기로 한다. 프로그램 1의 arg.c가 확인을 위한 프로그램이고, 화면 1이 실행한 예이다. 화면 1에서 알 수 있는 것처럼, main 함수의 인수, argc, argv에 각각 공백문자로 구분된 명령어의 인수 갯수와 인수자신이 넘겨지고 있기 때문에 프로그램측에서는 argv의 내용을 해석하여, 목적에 따른 처리를 하면 된다는 것을 쉽게 알 수 있다.

이와같이, C의 초기화루틴이 인수를 미리 처리하여 넘기는 것에 따라 두가지 장점이 있다. 하나는 UNIX와의 호환성을 갖고 있기 때문에 이식이 쉽다는 것이다. 그리고 또 하나의 장점은 위에 열거한 것과 같은 인수의 처리를 프로그램에서 명시적으로 할 필요가 없어 프로그램의 작성이 간단하게 되고, 또 프로그램의 보수성도 좋게 된다는 것이다.

명령어라인의 해석

C의 초기화루틴이 하는 처리는, 명령어라인을 분해하여 와일드문자의 전개를 하는 것이고, 그 내용을 어떻게 취급할 것인가는 프로그램에 맡겨져 있다. 명령어라인의 해석루틴을 포함하여, 3월호에서 설명한 규격을 따라 명령어 typen을 작성하여 보았다. 프로그램 2에 작성한 typen.c 프로그램을 typen 명령어를 사용하여 보였고, 화면 2에는 실제의 컴파일, 링크 그리고 실행예를 보였다. 명령어라인 typen의 핵심은 typen 함수 안에 기술되어 있다.

이 유틸리티 프로그램은, C의 원시 프로그램과 같은 텍스트에 행번호를 붙이고 싶은 경우 등에 매우 유용하다. 사실 본 연재에서 게재한 또 게재할 모든 프로그램 리스트는 해설의 편의상 행번호가 붙어 있지만, 말할 필요도 없이 프로그램 작성시에 행번호를 붙인 것이 아니라 이 typen 명령어을 이용하여 행번호를 붙인 것이다. 어느 곳으로 출력할 것인가를 지정하지 않으면, 표준출력 즉, 화면으로의 표시가 수행되지만, MS-DOS의 리다이렉트 기능을 이용하면, 출력결과를 화일에 출력한다든지, 혹은 프린터 및 그 이외의 디바이스 등에 출력할 수 있다. 예를 들면, text라고 하는 화일의 내용에 행번호를 붙여, text2라는 화일 이름으로 출력하고 싶은 경우에는 다음과 같이 입력하면 된다.

```
typen text > text2 ↵
```

또, 결과를 프린터로 출력하고 싶은 경우에는 다음과 같이 입력하면 된다.

```
typen text > prn ↵
```

〔그림 3〕 프로그램 2 typen.exe의 PSP 내용의 실예

```
C>symdeb typen.exe typen.c↵ ←— 심볼릭디버거를 기동하여 PSP의 내용을 본다.

Microsoft (R) Symbolic Debug Utility  Version 4.00
Copyright (C) Microsoft Corp 1984, 1985.  All rights reserved.

Processor is [80286]
-d0 1100↵  ←— 256바이트의 내용을 덤프한다.
 ┌→ PSP의 세그먼트 주소를 표시한다
16AF:0000  CD 20 00 A0 00 9A F0 FE-1D F0 8E 09 AB 0D 2B 0A  M . ..pp..+...
             └→ INT 20H
16AF:0010  AB 0D 56 09 AB 0D 9B 0D-01 03 01 00 02 FF FF FF  +.V.+...........
16AF:0020  FF FF FF FF FF FF FF FF-FF FF FE FF A6 16 82 8F  ............&...
16AF:0030  AB 0D 14 00 18 00 AF 16-FF FF FF FF 00 00 00 00  +...../.........
16AF:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
  INT 21H ←┐    ┌→ RETF
16AF:0050  CD 21 CB 00 00 00 00 00-00 00 00 00 00 54 59 50  M!K..........TYP
                                                ←——
16AF:0060  45 4E 20 20 20 43 20 20-00 00 00 00 00 20 20 20  EN   C .....
           —— 제1FCB ——————————→   ←——
16AF:0070  20 20 20 20 20 20 20 20-00 00 00 00 00 00 00 00          ........
           —— 제2FCB ——————————→
16AF:0080  09 20 74 79 70 65 6E 2E-63 20 0D 73 0D 79 70 65  . typen.c .s.ype
           └→ 인수의 문자수              └→CR코드:이후 옵세트FFH까지 무의미
16AF:0090  6E 2E 63 20 0D 73 63 35-3B 5C 62 61 74 63 68 3B  n.c .sc5;\batch;
16AF:00A0  0D 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
16AF:00B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
16AF:00C0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
16AF:00D0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
16AF:00E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
16AF:00F0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
-q ↵
C>
```

〔프로그램 1〕 arg.c 명령어 리스트

```
/* arg.c : 초기화루틴의 인수처리 확인 프로그램 */

 include <stdio.h>

main(argc, argv)
int  argc;
char **argv;
{
 register int i;

 printf("argc = %d,\n",argc);
 for (i = 0; i<argc; i++)
         printf("argv[%d] = %s, \n",i,argv[i]);
}

/*  end of arg.c  */
```

〔화면 1〕 arg 명령 실행화면

```
C>cl arg.c↵  ←— 전회에서 설명한 대로 컴파일러 드라이버 CL을 기동한다.
Microsoft (R) C Compiler Version 4.00
Copyright (C) Microsoft Corp 1984, 1985, 1986.  All rights reserved.

arg.c  ←—  컴파일 중임을 나타낸다.
Microsoft (R) Overlay Linker  Version 3.51
Copyright (C) Microsoft Corp 1983, 1984, 1985, 1986.  All rights reserved.

Object Modules [.OBJ]: ARG.OBJ    ┐
Run File [ARG.EXE]: ARG.EXE/NOI   │ 각 옵션에 대해
List File [NUL.MAP]: NUL          │ 디폴트로 수행된다.
Libraries [.LIB]: ;               ┘

C>dir arg.exe↵  ←—  실행형 화일이 만들어졌는지 확인한다.
 Volume in drive C has no label
 Directory of  C:\

ARG      EXE     6408   2-15-89   10:28a ←— 실행가능화일이 만들어졌다.
        1 File(s)   16211968 bytes free

C>arg↵  ←— 인수없이 arg 명령을 실행해 본다.
argc = 1 ,
argv[0] = C:\ARG.EXE,  ←— argv[0]에는 Full Path의 실행형 화일이름이
                           들어있다.

C>arg 1 2 3 4↵  ←— 단순한 인수 "1 2 3 4"를 지정해본다.
argc = 5 ,
argv[0] = C:\ARG.EXE,
argv[1] = 1,  ┐
argv[2] = 2,  ├— 인수들이 공백에 의해 구분되어 있다.
argv[3] = 3,  │
argv[4] = 4,  ┘

C>arg -d arg.c↵  ←— 옵션과같은 인수들의 처리는 어떤가 살펴본다.
argc = 3 ,
argv[0] = C:\ARG.EXE,
argv[1] = -d,
argv[2] = arg.c

C>arg *.c↵  ←— 인수에 와일드 문자가 들어있는 경우를 본다.
argc = 2 ,
argv[0] = C:\ARG.EXE,
argv[1] = *.c  ←—  SETARG.OBJ를 링크하지 않았기 때문에 와일드문자가
                   전개되지 않는다.

C>arg "*.exe"↵  ←—  '"'문자로 인수를 구분하여 지정해 본다.
argc = 2 ,
argv[0] = C:\ARG.EXE,
argv[1] = *.exe  ←—  '"'는 소거되고, 와일드문자는 그대로 넘겨진다.
```

〔화면 2〕 typen 명령어의 기동

```
C>cl typen.c↵  ←— 컴파일러 드라이버 CL을 기동한다.
Microsoft (R) C Compiler Version 4.00
Copyright (C) Microsoft Corp 1984, 1985, 1986.  All rights reserved.

typen.c
Microsoft (R) Overlay Linker  Version 3.51
Copyright (C) Microsoft Corp 1983, 1984, 1985, 1986.  All rights reserved.

Object Modules [.OBJ]: TYPEN.OBJ
Run File [TYPEN.EXE]: TYPEN.EXE/NOI
List File [NUL.MAP]: NUL
Libraries [.LIB]: ;

C>dir typen.exe↵  ←—  실행형 화일이 만들어졌는지 확인한다.
 Volume in drive C has no label
 Directory of  C:\

TYPEN   EXE     8356   2-16-89  11:15a ←— 실행가능화일이 만들어졌다.
        1 File(s)   16211968 bytes free

C>typen typen.c↵  ←—  자신의 텍스트화일에 문번호를 붙여본다.
## <프로그램2> ##
```

프로그램 2의 리스트를 보자. C언어의 프로그램은 하나 이상의 함수의 집합체이다. typen.c의 경우에는, main이라는 이름의 함수(이것이 주 루틴이다)와, typen이라는 이름의 함수의 합계 2개의 함수로 구성되어 있다. 프로그램 리스트 중에서, 앞부분 12-33행이 주(main)루틴이고, 뒷부분 35-50행이 부(sub) 루틴(함수)이다. 그러나 주 루틴에서 수행하는 것은, 엄밀히 말해 처리의 핵심부는 아니다. 다만 화일을 오픈하여 함수를 실행하고, 화일을 클로즈하는 것 뿐인 처리이다. 실제로 화일을 읽어낸다든지, 그것에 행번호를 붙인다든지, 출력한다든지 하는 처리는 뒷부분에 맡겨져 있다.

C언어에서는, '/＊'와 '＊/'사이에 있는 부분은 코멘트로 간주되기 때문에, 컴파일러는 이 부분을 무시하고 컴파일을 수행한다. 또한 이 부분은 프로

〔프로그램 2〕 텍스트 화일에 문번호를 주는 프로그램

```
0001: /*
0002:  *    텍스트에 라인번호를 부여하는 프로그램
0003:  *
0004:  *                          작성자 : 이 용 일
0005:  */
0006:
0007:  2nclude <stdio.h>
0008:
0009: extern void main ( int, char ** );
0010: extern void typen ( FILE * );
0011:
0012: void main ( argc, argv )
0013:
0014:        int argc;             /* 인수의 갯수 */
0015:        char *argv [];        /* 인수 벡터   */
0016: {
0017:        FILE *fp;             /* 입력화일에 대한 화일 포인터
0018:
0019:        if ( argc == 1 ) {
0020:            typen ( stdin );
0021:        }
0022:        else {
0023:            if ( ( fp = fopen ( *++argv, "r" ) ) == NULL ) {
0024:              fprintf(stderr,"typen: cannot open %s\n",*argv);
0025:              exit(1);
0026:            }
0027:            else {
0028:                    typen (fp);
0029:                    fclose (fp);
0030:            }
0031:        }
0032:        exit(0);
0033: }
0034:
0035: void typen (fp)
0036:
0037:        FILE *fp;             /* 입력파일에 대한 화일 포인터
0038: {
0039:        int num = 1;          /* 행번호 */
0040:        int c;                /* 문자 (character) */
0041:
0042:        while ( ( c = fgetc (fp) ) != EOF ) {
0043:                printf ( "%04d: ", num++ );
0044:                while ( c != '\n' && c != EOF ) {
0045:                        putchar (c);
0046:                        c = fgetc (fp);
0047:                }
0048:                putchar ( (int) '\n' );
0049:        }
0050: }
0050: }
```

그램의 실행에 어떠한 영향도 주지 않는다. 7행에서 stdio.h라는 화일을 포함(include)하고 있다. 대부분의 C언어 처리계에서는 이 include를 써놓지 않으면, fopen / putchar / fgetc / fclose 등의 표준적인 입출력함수를 사용할 수 없다. C언어 프로그램의 작성에서 정석과 같은 것이라 이해해도 무리가 없을 것이다. 9-10행에서, m ain과 typen의 형을 선언하고 있다.

이 선언을 프로토타입 선언이라 하며, 현재의 마이크로소프트 C 버전3.0이상에서는 (래티스 C 버전3.0 이상) 이 선언을 지원하고 있지만, 이전의 버전 및 그외의 컴파일러에서는 프로토타입 선언이 수행되지 않는 경우도 있다. 사전에 프로토타입 선언을 해 놓으면 컴파일할 때, 컴파일러가 프로그램의 오류(함수를 호출하는 측과 호출되는 함수의 형의 불일치)를 지적하여 주기 때문에 큰 도움이 된다.

표준함수(라이브러리)에 관해서는 화일의 이름 중 확장자가 '.h'인 헤더화일안에 프로토타입 선언이 되어 있다. 따라서 적절한 헤더화일을 포함(include)해 두면, 다시 프로토타입 선언을 할 필요는 없다. 12-15행의 선언에 의해 함수 main(주 루틴)이 시작된다. 12행 main뒤의 괄호안에 2개의 인수가 기술되어 있기 때문에, 이 함수 main이 2개의 인수를 받는다는 것을 알 수 있다.

C언어의 주 루틴이 받는 2개 인수의 의미는, 언어 규격상 정해져있는 것이다. 첫번째 인수 argc는 DOS의 명령어 라인에 기술된 단어의 갯수이다. 두번째 인수 argv는, 각각의 단어가 격납된 배열변수의 선두주소(포인트)가 격납되어 있는 배열변수의 선두를 가리키고 있다. 조금 구체적인 예를 들어 설명하면, 만일 MS-DOS의 명령어라인으로서, 다음의 한 명령어라인이 입력되었다고 하자.

```
typen 123 abcd 456 ↵
```

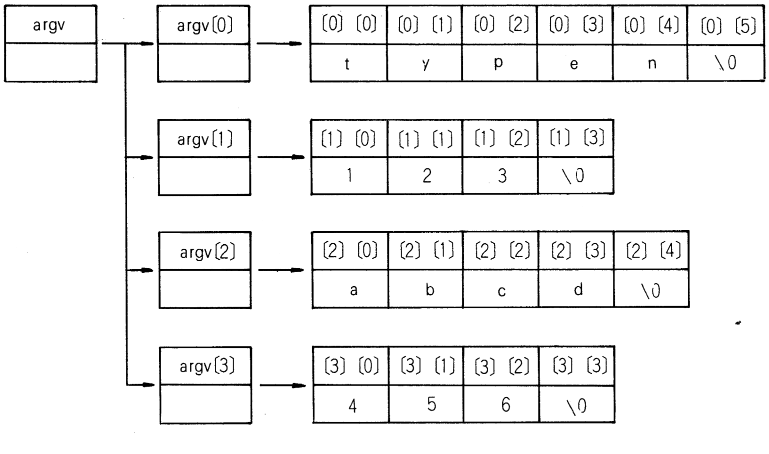
이때 typen 프로그램의 main이 받아들이는 최초의 인수 argc에는, 단어의 수가 4개이므로 4가 들어간다. 기동한 프로그램 자신도 갯수에 포함되는 것에 주의를 기울여야 한다. 또 argv에 주어진 값(포인트)을 해석하는 것에 의해 그림 4에 보이는 것 처럼, 4개의 단어를 알 수 있게 되어 있다. 즉, argv[0]는 문자열 'typen'으로의 포인터, argv[1]은 문자열 '123'으로의 포인트, argv[2]는 문자열 'abcd'로의 포인터, argv[3]은 문자열 '456'으로의 포인터가 된다. 다만, MS-DOS의 제약 때문에 버전 2.XX 이전의 MS-DOS용으로 릴리즈된 처리계에서는, 최초의 단어인 기동된 프로그램 자신은 할 수 없다.

17행에서 화일 포인터 fp를 사용하는 것을 정하고 있다. C언어에서 고수준의 화일을 조작하려면 하나의 화일을 오픈하는데 대해 하나씩의 화일 포인터가 필요하다. typen.c는 행번호를 붙이는 원시 프로그램 또는 텍스트를 화일로부터 읽어내기 때문에, 그것을 위한 화일포인터를 여기에서 확보하고 있다. 19-21행에서는, 인수 argc의 값이 1이라는 것은, MS-DOS의 명령어 라인에는 기동하는 프로그램 이름 밖에 기술되지 않았다는 것, 즉 화일 명의 지정이 없었다는 것을 의미하고 있게 때문에 텍스트를 화일로부터 읽어내는 것이 아니라 표준입력에서 읽어내게 된다.

보통 C언어에서는 표준입력에 대한 화일 포인터는 stdin이라는 명칭으로 자동적으로 오픈되고 있기 때문에, 그 화일 포인터를 오픈할 때의 화일 포인터로서 부여하여 함수 typen을 호출하고 있다. C언어 처리계에서는, stdin이외에도 몇개의 화일 포인터가 자동적으로 오픈되고 있으며, 표 1에 그들의 화일 포인터와 디바이스의 대응을 보이고 있다.

인수 argc의 값이 1 이외인 경우, 즉 하나 이상의 화일명이 지정된 경우에

〔그림 4〕 명령어 인수의 획득



〔표 1〕 화일 포인터와 디바이스

화일 포인터	디바이스
stdin	표준 입력(키보드)
stdout	표준 출력(CRT)
stderr	표준 에러(CRT)
stdaux	보조 입출력(회선)
stdprn	프린터

는 22-32 행을 실행한다. 먼저, 23행과 같이, 지정된 이름의 화일을 오픈한다. 이때 fopen을 실행한다. 이 함수는 화일이 존재하지 않는 등의 어떤 이유로 정상적으로 화일이 오픈되지 않았을 때 NULL을 돌리기 때문에, 이것이 NULL 이면 에러메시지를 stderr에 출력하고 처리를 중단한다. 지정된 화일이 정상적으로 오픈되면 오픈한 경우의 화일 포인터 fp를 넘겨 함수 typen을 실행한다. 그리고 fclose로 화일을 클로즈한다.

입력이 stdin이든, 화일이든, 정상적으로 처리가 종료되면, 32행으로 제어가 옮겨진다. 보통 exit를 호출하는 것에 의해 프로그램을 종료하지만 이때 정상 종료를 의미하는 '0'을 주어 이 함수를 호출하고 있다.

함수 typen은, 이미 오픈된 화일의 화일 포인터를 넘겨받는다. 그 화일로부터 텍스트를 읽어내고 행번호를 붙여 표준출력으로 출력하는 것이 이 함수의 목적이다. 화일 포인터는 물론, 보통의 디스크화일에 대응하는 포인트라도 상관없다. 또, 표준입력의 화일 포인터 stdin을 받으면 키보드로부터 입력된 텍스트에 대해, 행번호를 붙여 표준출력으로 출력한다. 만일, 주 루틴에서 보조입력의 화일 포인터 stdaux를 넘겨받으면 통신회선에서 입력된 텍스트에 행번호를 붙이는 것도 가능하다.

35-37행에서는 함수 typen을 정의한다. C언어에서 함수의 정의는 먼저 함수의 이름을 기술하고 그뒤의 괄호안에 인수의 이름을 나열하는 것이다. 주 루틴도 하나의 함수이기 때문에 이 정의는 주 루틴인 main의 경우에도 함께 적용된다. 프로그램 2의 프로그램 리스트로부터 함수 typen은 단 하나의 인수 fp를 주 루틴으로부터 넘겨받는 것을 알 수 있다. 인수의 이름까지는 알 수 없지만 인수의 형 만이면, 10행의 프로토타입 선언에서도 알 수 있다.

fp의 형은 'FILE형'이므로 함수 typen이 받아들이는 것은 화일 포인터이다. FILE형이 실제로 어떤 것인가는 stdio.h안에 정의되어 있지만, 화일을 액세스할 때의 정석으로서 프로그램 리스트를 보아두면 그 내용까지 의식할 필요는 없다. 받아들인 인수에 대해서는,

그 값만을 사용할 수 있다. 이것을 'call by value'라고 한다.

함수중에서 참조하는 fp는, 그 형 및 값은 원래의 것(주 루틴에서의 fp)과 같지만, 실제로는 복사물일 뿐이다. 그것이 베이직 등의 변수 혹은 글로벌변수 등과는 다른 개념이다. 예를 들면 함수안에서 fp의 값을 변경하는 것은 불가능하지만, 그 값은 주 루틴에 돌아가는 시점에서, 함수가 실행되기 전의 값으로 원상복귀한다. 직접 값을 변경하고 싶으면, 그 값이 격납되어 있는 장소(포인트)를 넘겨, 그 포인트를 참조하여 값을 변경하여야만 한다.

39-40행에서는 함수 typen안에서의 2개 변수, 즉 num과 c를 사용한다는 것을 선언하고 있다. 둘다 정수형의 변수이므로, -32768부터 +32767 사이의 값을 취급할 수 있다. 39행에서는 선언과 동시에 변수 num에 행번호의 초기화로서 1을 대입하고 있다. 42행은, 49행과의 중괄호로 둘러싸인 부분을 반복하기 위해 while을 사용한 루프를 만들었다. while에 의한 반복처리는 , while 뒤의 괄호안에 조건식이 틀릴때까지 반복되는 것이다. 즉 fgetc를 사용하여, 화일로부터의 입력이 화일 끝인 EOF에 도달할 때까지 반복처리하는 것이다. 그와 동시에, 42행에서는 화일로부터의 입력문자를 변수 C에 격납하고 있다.

표준입력으로부터의 입력인 경우에는 'CTRL-Z' 키가 입력된 경우가 EOF가 된다. 43행에서는 행번호를 출력한다. 함수 printf의 인수로서 주어진 최초의 문자열은 출력의 양식을 지정한다. 여기서는 정수값을 4자리수로 출력하고, 그뒤에 ':'과 공백을 부가하여 출력할 것을 지정하고 있다. 행번호의 출력과 동시에 '++' 연산자를 사용하여 행번호인 변수 num의 값에 1을 더하고 있다.

44-47행의 반복처리에 의해 한 행의 데이터를 화일 혹은 표준입력등에서 읽어내어 그것을 그대로 표준출력하고 있다. 읽어내는 데는 fgetc를, 출력하는 putchar함수를 이용한다. 반복처리를 종료하는 것은 화일로부터 화일끝 표시인 EOF 또는 행끝 표시인 '\n'이 입력되었을 때이다. 48행은 한 행의 출력을 종료하여 44-47행의 반복처리를 마치고나서 행끝 표시인 '\n'을 표준출력으로 출력해두는 것이다. 그리고 다시 42행으로 돌아가 다음의 행을 읽어내고 출력을 반복한다.

마이크로프로세서 마이크로컴퓨터 講義案內 개강 매월 5일 (2개월 완성)

(하드웨어 과정)

본원에서는 금번 2개월 과정으로 digital circuit 설계로 부터 CPU 및 memory, 주변장치 등의 구조 및 동작원리를 습득하여 computer system 및 주변기기의 Interface를 설계하는 능력을 함양하고자 합니다. 또 이 과정에서는 Z-80계 computer 및 APPLE, IBM PC 등의 system을 대상으로 구성, 동작원리, 프로그래밍 실습을 하여 궁극적으로 MICOM 및 digital system의 설계 및 수리능력을 배양하고자 하오니 많은 참여 바랍니다.

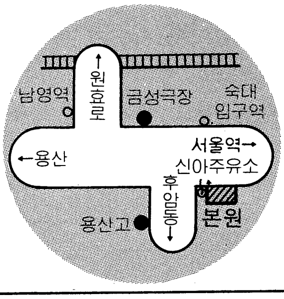
■ 강습일정

주별	일차	강습 내용 (이론 및 실기)
1 주	1 일차	computer의 기본 구조와 8bit CPU의 종류 및 내부 구성
	2 일차	논리회로 설계원리 및 그 응용
	3 일차	마이크로 컴퓨터 시스템의 구성
2 주	1 일차	Karnaugh Map을 이용한 디지탈회로 설계
	2 일차	Flip Flop 및 Counter 등을 이용한 응용회로 설계
	3 일차	CPU의 주변회로 및 Address Map/Decoding
3 주	1 일차	TTL 논리회로 설계 실습
	2 일차	A/D 및 D/A Converter 및 ALU
	3 일차	Memory(ROM/RAM/DRAM)의 구성
4 주	1 일차	8080 및 8085 CPU의 구조 및 Timing 특성
	2 일차	8085 Assembly언어 프로그래밍
	3 일차	I/O(Port, Keyboard etc)의 원리 및 Video Generator
5 주	1 일차	Z80 CPU의 구조 및 Timing 특성
	2 일차	Z80 CPU의 Interrupt (IM0, IM1, IM2)
	3 일차	μ-Computer의 회로설계 및 실습
6 주	1 일차	Assembly Language를 이용한 프로그램 실습
	2 일차	8255 및 Z80 P10의 특성 및 응용
	3 일차	16bit MPU(MC68000)의 구조 및 원리
7 주	1 일차	Z80 S10의 특성 및 응용
	2 일차	EPROM Writer 및 Floppy Disk Controller
	3 일차	MC68000의 Signal 및 Bus Operation
8 주	1 일차	음성합성 장치의 설계
	2 일차	Bus Standard, Modem, Data통신 소개
	3 일차	16bit MPU(MC68000)의 회로 설계

Textbook

1. TTL응용실무
2. Z80 마이크로컴퓨터
3. Microprocessor Interfacing Technique
4. Microprocessor and Digital Systems

Hardware + Software = ?



전자산업 4반세기를 지켜온 기술인의 전당

인가 **한국전자통신학원** 컴퓨터 기술부

(구 한국전파학원)

140 서울시 용산구 갈월동 17번지 ☎ 754－1751/2902/1044

미분이 가능한 계산기 프로그램

조성만/단국대학교 전자공학과

여기에 소개하는 프로그램은 터보파스칼을 이용하여 작성되었으며, 공학도들이 필요로 하는 여러가지 복잡한 계산을 처리할 뿐만 아니라 단순한 미분만이 아닌 여러가지 복잡한 수식의 처리도 간단하게 해결할 수 있다.

필자는 공학을 전공하고 있어서 수식계산을 많이 하여야 하는 편이다. 그러나 대부분 단순한 형태의 것들이어서 휴대용 계산기를 이용하여 계산을 하지만 가끔 계산량이 많다 싶은 것들은 컴퓨터를 이용하기도 한다.

필자가 주로 사용하는 언어는 파스칼인데 독자 여러분들도 잘 알다시피 파스칼에서는 지수승의 계산을 할 수 없다. 별수없이 지수승을 계산하려면 파스칼 컴파일러내에 내장되어 있는 함수들을 조합한 형태로 변환시켜 주어야 한다.

예를 들면 a^b은 exp(b∗ln(a))와 같이 표시해야 한다. 이와 같은 과정은 분명히 번거로운 작업임에 틀림없다. 또한 수학적 배경이 적은 사람은 이와 같은 공식을 자주 사용하지 않기 때문에 a^b=exp(a∗ln(b))인가? a^b=exp(b∗ln(a))인가? 하는 질문에 금방 자신있게 대답할 사람은 별로 없을 것이다.

또한 전산학에 관심이 있어 수치해석 과목을 배워 본적이 있는 사람이라면 근의 값을 구하는 과정중에 뉴우톤 랩슨의 방법을 이용하여 근을 구하는 프로그램을 작성해 본 경험이 있을 것이다. 이 과정중에 미분식을 이용하는 부분이 있는데 다행이 미분하려는 식이 간단한 것이라면 별문제가 없겠으나 지수함수나 삼각함수 등으로 된 복잡한 식일 경우에는 수학책을 다시 펴서 공식을 찾고, 이를 이용하여 원래의 식을 미분하고 미분이 제대로 되었나 확인해 보아야 하는 번거로운 작업을 해야 할 것이다.

이러한 이유로 필자는 약간의 자료를 정리하고 프로그래밍하여 공학도들에게 풍부한 수학적함수 라이브러리를 제공하여 자신의 프로그램에 사용할 수 있게 하는 동시에 수학적 배경이 적은 비공학도들에도 사용할 수 있게끔 단순화시킨 계산기 프로그램을 소개하고자 한다.

계산기 프로그램

이 프로그램은 애플용 터보 파스칼 버전 2.0에서 작성되었으며, 프로그램은 몇개의 모듈로서 분리되어 있어 필요한 모듈들을 조합하여 하나의 완벽한 프로그램으로 만들수 있으며, 기본적인 함수들을 제공하는 func에서 필요한 루틴들만을 따로 떼어내어 사용할 수도 있다. 그러므로 사용자가 데이터 및 프로그램의 표준 라이브러리처럼 사용할 수 있도록 하였다.

구성을 보면 크게 일반적인 연산의 결과값을 계산하는 부분과 미분식을 계산하는 미분계산 두부분으로 나눌 수 있는데 거의 완벽한 미분식을 얻을 수 있는 특징을 갖는다.필자는 여기에서 함수계산부분인 func부분을 간단히 설명하고 중요한 미분계산기(Differential Calculator) 프로그램을 위주로 설명하겠다. 그리고 이 프로그램의 이후의 명칭을 Calc로 하겠다.

기본적인 수학함수공식(Func)

Func에는 **리스트 6**에 보이는 바와 같이 터보 파스칼내에 정의되지 않은 삼각함수 및 쌍곡선함수들을 계산할 수 있는 함수들이 준비되어 있다. 이외에도

- log(x) : log x 값을 구하여 준다.
- rad(x) : x 값을 디그리로 보고 라디안값으로 변환시켜 준다.
- deg(x) : x 값을 라디안에서 디그리로 변환시켜 준다.
- rmod(x, y) : x mod y
- rdiv(w, y) : x div y
- power(a, b) : a^b
- facto(n) : n!
- Perm(a, b) : ${}_aP_b$←순열의 수
- Comb(a, b) : ${}_aC_b$←조합의 수

등 약 30여개의 함수들이 준비되어 있어서 기존의 내장함수와 같이 사용할 수 있다. 또한 이 부분은 독자여러분이 프로그램을 작성할 경우 필요한 루틴을 따로 떼어내어 사용할 때 약간만 수정하여 사용한다면 매우 유용한 루틴이 될 수 있다.

예를 들어 Power를 사용할 경우 errorflag(에러번호)를 writeln(에러내용)만 고쳐주면 된다(리스트 5 참조).

func.pas는 이 함수들이 호출되었을 때 에러체크를 하게 되는데 이때 "asin(3)"과 같이 잘못된 범위의 값(arc sin(x)에서 X의 값은 abs(x)≦1.0이어야 한다.)을 지정했을 경우에는 프로시듀어 에러플래그(errorflag)를 호출하여 변수 에러를 True로 세트하고 변수 에러 코드에 그에 해당하는 에러번호를 부여하게 된다.

이 에러코드의 값은 다시 에러플래그(0)가 호출되기 전까지는 변경되지 않는다. 독자여러분들은 프로시듀어 에러플래그를 자신들의 개성에 맞게 수정하거나 내용을 추가하면 좀 더 유용한 에러처리 루틴이 될 것이다.

다음 리스트 1은 Func를 터보 파스칼 4.0에서 사용한 예이다.

〔리스트 1〕

```
PROGRAM Sample_1 ;
uses func ;

begin

  writeln('*****  FUNCTION TEST  *****') ;

  writeln;

  writeln(' TAN(0.5) : ', tan(0.5):7:3   ) ;
  writeln('ASIN(0.5) : ', asin(0.5):7:3  ) ;
  writeln('SINH(0.5) : ', sinh(0.5):7:3  ) ;
  writeln('ATAN(0.5) : ', atan(0.5):7:3  ) ;
  writeln(' 5 ^ 3    : ', power(5,3):7:3 ) ;
  writeln(' 5 !      : ', facto(5):7:3   ) ;
  writeln(' 5 P 3    : ', perm(5,3):7:3  ) ;
  writeln(' 5 C 3    : ', comb(5,3):7:3  ) ;

end.
```

미분계산기의 분석

(1) 변수 및 함수

Calc 프로그램중에 사용된 중요 변수 및 함수는 다음과 같다.

① 데이터 정의부(Type)

• Polyhead : Calc에서 사용되는 포인터형태의 기본 데이터형으로 이 변수로 선언되면 각각 하나의 항을 나타내며 여러 개의 데이터들이 세가지형태의 연결 포인터로 연결되어 전체의 수식을 구성한다.

② 전역변수

• TabConst : 숫자를 문자열로 바꾸어 줄 때 소숫점이하 몇째 자리까지 나타내 줄것인가 결정하게 된다.(초기치 TABCONST := 2)

• error : 연산 혹은 실행중에 에러가 발생하였을 때 true로 세트하며 초기치는 false이다.

• errorcode : 에러가 발생하였을 때 그에 해당하는 에러의 번호를 가지게 되며 에러가 없으면 0을 나타낸다.

• Varname : Calc가 미분할 때 대상이 되는 변수이다. 예를 들면 X^3+2XY와 같은 식을 'X'로 하느냐 'Y'로 하느냐 지정해 주어 편미분한 결과값을 준다. 초기치는 X이다.

• Processflag : 미분할 때 중간과정을 보여줄 것인가를 결정한다. 초기치는 False이다.

• dgtbox : Calc에 준비된 compute 기능을 이용하여 결과값을 계산할 때 dgtbox에서 그에 해당하는 값을 참조하여 계산한다. 예를 들면 3a+4x는 dgtbox ['a']와 dgtbox['x']의 값을 참조하여 각각 1, 0일 경우에 결과값 3을 계산한다.

③ 기본함수

• Makehd : 문자열 형태의 수식의 입력을 Calc에서 사용하기 위한 새로운 자료형으로 재구성해 준다.

• Copyhd : Makehd에 의해 만들어진 자료를 복사하여 메모리내에 똑같은 자료를 만든다.

• Erasehd : Makehd에 의해 새로운 자료형이 만들어지면 메모리내에 일정영역을 차지하게 되며 이 동작이 반복되면 메모리의 모든 영역을 차지하게 된다. 이때 이 함수를 사용하면 Makehd에 의해 점유된 메모리를 해제하게 되며, 메모리내에 자유영역을 만들어 주어 계속적인 작업을 행할 수 있게 해 준다.

• Computehd : Makehd에 의해 만들어진 수식을 이용하여 결과값을 계산해 준다.

• Diffhd : Makdhd에 의해 만들어진 수식을 미분하여 메모리내에 미분된 수식을 만들어 준다.

• Adjust : Makdhd, diffhd에서 만들어진 수식을 제조정하여 보다 정리된 형태의 수식으로 변형시켜 준다.

• Convert : Makdhd, Diffhd에 의해 만들어진 수식을 문자열 형태로 바꾸어 준다.

④ 응용함수

• Diff : 앞의 기본 함수들을 조합하여 만들어진 것으로 문자열형태의 수식을 입력받아 이를 미분한 후 미분된 문자열로 출력한다.

• Compute : 기본 함수들을 조합하 여 문자열 수식을 입력받아 이를 계산한 후 결과를 출력한다.

• errorflag : Calc에서는 각각의 루틴을 처리할 때 에러체크를 한다. 만약 실행도중 에러가 발생하면 프로시듀어 에러 플래그를 호출하여 에러변수를 true로 하고 에러코드에 해당하는 숫자값을 갖게 된다. 사용자는 이 에러플래그를 수정하면 좀

더 효율적인 에러처리 루틴을 만들 수 있을 것이 다.

(2) 모듈

Calc는 여러개의 모듈로 분리되어 있으며 각 모듈들은 여러개의 프로시듀어 및 함수들로 구성되어 있다. 그러면 Calc의 전체중에서 문자열 형태의 수식을 미분하여 결과를 얻기까지 필요한 모듈의 대략적인 기능을 알아보자.

- Declare.inc : 프로그램 중에 사용되어질 상수, 타입, 변수 등이 선언되어 있다.
- Util.inc : 프로그램중에서 공통적으로 사용되어질 각종 서브루틴 형식의 프로시듀어 및 함수들의 모임으로 func.pas와 같이 이 부분에서 필요한 부분만 따로 떼어 사용할 수 있어 여기서 제공되는 함수를 편리하게 이용할 수 있다.
- Make.inc : 문자열 형태의 수식을 입력받아 포인터를 이용하여 메모리내에 새로운 형태의 자료구조형으로 재구성시킨 다음 다른 형태의 연산(계산이나 미분)을 행할 수 있도록 준비시켜 준다. 이후부터는 이 포인터를 이용하여 재구성된 자료구조형을 폴리노미얼(Polynomial)이라 칭하겠다.
- Copy.inc : 포인터형의 폴리노미얼을 복사하여 메모리내에 똑같은 형태의 새로운 폴리노미얼을 만들어 준다.
- diff.inc : 포인터형의 폴리노미얼로 변환된 수식을 미분하여 미분된 다른 폴리노미얼을 생성시켜 준다.
- Adjust.inc : 폴리노미얼 형태의 수식을 재조정하여 필요없는 괄호 및 '0'을 포함한 항 등을 정리하여 보다 정리된 형태의 폴리노미얼로 만들어 준다.
- Convert.inc : 폴리노미얼 형태의 수식을 문자열 형태의 수식으로 변형시켜 준다. Make.inc의 역변환이라 할 수 있다.

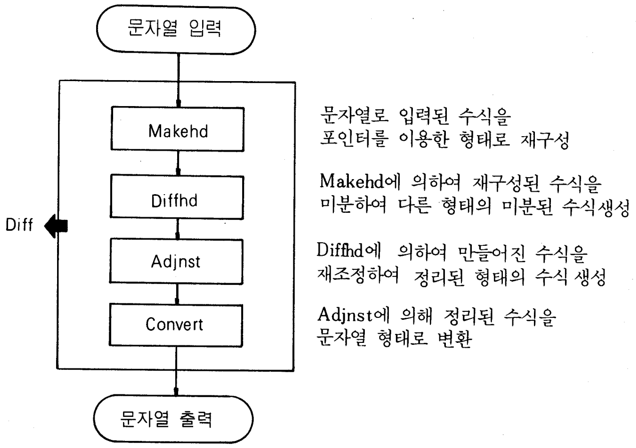
(3) 동작방법

Df는 **그림 1**과 같이 문자열 형태의 수식을 입력받아 이를 그대로 미분할 수 없으므로 포인터를 사용한 폴리노미얼 형태로 재구성시킨 다음 연산을 행하고 이를 다시 문자열 형태로 바꾸어서 출력하여 준다. 전체 소스리스트는 **리스트 8**에 나타나 있다. 단, 여기서 모듈별로 작성되어 있다는 것을 염두에 두길 바란다.(앞에서 설명한 Calc는 전체 프로그램을 말하며 Df는 Calc중에서 미분식을 얻기위해 필요한 모듈들의 집합임을 참고하기 바란다)

이와 같이 Df가 자동적으로 미분을 행하여 주므로 사용자는 Df를 블랙박스처럼 취급하여 단지 문자열형태의 수식을 정리하여 Diff('4X^2+3X+5')와 같이 호출하면 된다. 그러면 Df는 미분된 결과인 8X+3과 같이 문자열 형태로서 결과값을 돌려 준다(**리스트 2 참조**).

Df는 별다른 지시가 없는한 모든 수식을 X에 대하여 미분을 행한다. 그러나 Df는 변수 X뿐만 아니라 'A-Z'까지의 모든 변수에 대하여도 미분이 가능하다.(단, 문자 C와 P는 조합의 수와 순열의 수를 구하는 함수명으로 쓰이고 있으므로 불가능하다.)

〔그림 1〕 Diff의 처리과정



〔리스트 2〕

```
Program Sample_2;
(*$I Df.inc *)
var Y : string[255];

begin
  init;
  Y := Diff('4x^2 + 3x + 5');
  writeln('Result : ', Y);
end.
```

〈결과〉

```
Result : 8X+3
```

그러므로 만일 독자들이 Df를 이용하여 미분식을 얻고자 할때 별다른 지시없이 단순히 Diff('X^3+Y^2')와 같이하면 Df는 변수 X에 대해 편미분한 결과 값 3X^2+2*Y^2을 돌려 준다.

그러나 이 식을 Y에 관하여 미분하고자 할 경우에는

```
Varname := 'Y';
Y := Diff('X^3 + 2XY^2');
```

와 같이 하면 된다. 이때 반드시 변수 이름은 대문자로 지정해야 한다. 그러면 Df는 변수 y에 대하여 편미분한 결과값인 4XY를 결과값으로 돌려 보내준다.

그러나 주의할 점은 이와 같이 한 후에는 Df는 모든 함수를 Y에 대하여 미분을 행하게 된다. 다시 X에 대하여 미분을 하고자 하면

```
Varname := 'X'
```

와 같이 지정을 한 후에 사용하면 된다.

이와 같이 Df는 Varname로 지정된 문자에 대하여 편미분을 행한다. Varname의 초기치는 'X'로 지정되어 있으며 프로그램중에서 다른 문자로 변경가능하다.

Df는 이외도 일차이상의 고계도 함수 및 각종 여러가지의 응용을 할 수 있는데 이와 같은 조작을 하려면 약간의 수정 및 응용 루틴이 필요하므로 여기에서는 임의의 변수에 대해 미분을 행하는 단순한 예만을 소개하도록 하겠다(리스트 3 참조).

또한 필자는 Df가 미분을 행하는 과정중에서 미분의 중간과정을 보여주는 루틴을 추가하였는데 중간계산과정을 보고 싶으면 Diff를 호출하기 전에 변수 프로세스플래그(Processflag) 를 True로 세트하면 된다.

```
.
.
Processflag := True
Diff('3X^2 + 4X + 5');
.
.
```

이와 같이 하면 이후부터 Diff가 호출되며 식의 중간형태를 보여주게 된다. 그러나 이 중간과정을 보여주는 루틴은 임시로 만들어 놓은 것이므로 깔끔하게 정리된 형태의 중간과정을 보여주지 못한다. 여기에서 필자는 이 부분을 좀더 보완한다면 CAI 프로그램에도 이용할 수 있을 것이라고 예상하고, 이 부분에 관하여는 관심있는 독자에 의해서 좀더 정리된 형태로 개발되었으면 하는 바램이다.

다시 중간과정이 필요없을 경우에는 프로세스 플래그를 false로 하면 Diff는 중간의 계산 결과를 보여주지 않고 단지 완전히 미분된 결과만을 문자열/형태로 되돌려 준다(리스트 4 참조).

(4) 사용법

① 8비트에서의 사용

8비트에서는 모듈구조로서 include를 사용하였는데 이 include 문은(*$I declare.inc *)와 같이 사용하면 컴파일러는 declare.inc를 디스크로부터 읽어들여 이곳에 위치시킨 후 컴파일한다. 이것은 소스 리스트를 이곳에 적은 것과 같은 효과를 나타낸다. 다음 리스트에 그 사용방법을 나타낸다.

```
Program Bit8;
(*$I Declare.inc *)
(*$I Util.inc    *)
(*$I Make.inc    *)
(*$I Ajust.inc   *)
(*$I Convert.inc *)
(*$I Copyhd.inc  *)
(*$I Diff.inc    *)
Var Y:string[255];
begin
  init;                     {초기화 루틴}
  Y := Diff('3x^2 + 4x');  {함수 Diff를 호출하여 미분한 후 결과값을 Y에 저장}
  writeln(Y);               {결과값 출력}
end.
```

〔리스트 3〕

```
Program Sample_3;
(*$I Df.inc *)
var s, y1, y2, y3 : string[255];

begin
  init;
  s := 'x^3 + 2y^2';
  y1 := diff(s) ;
  varname := 'Y';
  y2 := diff(s) ;
  varname := 'X';
  y3 := diff(s) ;
  writeln('Y1 : ', y1);
  writeln('Y2 : ', y2);
  writeln('Y3 : ', y3);
end.
〈결과〉
Y1 : 3X^2
Y2 : 4Y
Y3 : 3X^2
```

〔리스트 4〕

```
Program Sample_4;
(*$I Df.inc *)
var s, y : string[255];

begin
  init;
  s := '4x^2 + 3x + 5';
  writeln('Now Processflag OFF');
  y := diff(s);
  writeln('RESULT : ', y);
  processflag := true;
  writeln;
  writeln('Now Processflag ON');
  y := diff(s);
  writeln('Result : ', y);
end.
```

그러나 8비트에서 사용할 경우 터보 파스칼 2.0의 에디터는 용량이 20KB 정도이므로 뒤에서 나타날 **리스트 8**번의 전체 리스트를 한꺼번에 작성할 수 없다. 그러므로 모듈별로 나누어서 작성하길 바란다. 그리고 **리스트 7**과 같은 프로그램을 작성할 경우 메인메뉴 상태에서 메인 화일의 이름을 지정해 주고, 옵션을 디스크 컴파일상태로 만들어 주고 컴파일하면 디스크에 컴파일된 화일이 생성된다. 프로그램의 실행은 도스상태에서 행하여야만 한다.

② 16비트에서의 사용법

16비트에서는 터보 파스칼 4.0에서 작성하여 사용할 것을 권하고 싶다. 이것은 8비트에서 작성한 부분에서 약간만 수정하면 될 뿐더러 터보 파스칼 4.0에서 제공하는 Unit구조를 사용하여 프로그램을 모듈화시켜 사용하면 다시 컴파일할 필요가 없기 때문에 사용이 간편하고 속도가 빠르다.

●Unit로 만드는법

```
Unit Df;
interface                      <--이곳에 나타난 상수, 변수, 타입, 프로시듀어,함수
(*$I Declare.inc *)                 등은 외부에서 참조 가능하다.
Function Diff(SS:s255):s255; <--외부에서 참조하고자하는 함수 및 프로시듀어들의
implementation                      헤드 부분만을 이와 같이 이곳에 모두 적는다.
(*$I Util.inc    *)                 여기서 SS는 파라미터이고 S255는 Declare.inc에서
(*$I Make.inc    *)                 Type S255 = string[255]라고 선언되어 있다.
(*$I Ajust.inc   *)
(*$I Convert.inc *)
(*$I Copyhd.inc  *)
(*$I Diff.inc    *)
begin
  init;  {Unit내에서 초기화루틴을 수행}
end.
```

이 프로그램을 디스크에 컴파일하면 화일명 Df.tpu가 디스크에 생성되며 이후에 다른 프로그램에서 이를 사용하고자 하면 다음의 예와 같이 단지 프로그램의 선두에서 USES Df와 같이 선언한 후에 사용한다. 그리고 앞에서 사용하였던 include화일과 비교하였을 때 Unit Df에서 초기화루틴을 수행하므로 초기화 루틴이 필요없다.

●사용예

```
Program Bit16;
USES Df;
Var Y:string[255];
begin
  Y := Diff('3x^2 + 4x');
  writeln(Y);
end.
```

(5) 특징

Calc는 그 특징으로 수학식의 입력 및 출력형태를 들 수 있다. Calc는 수학식의 입력 및 출력형태가 사람이 사용하는 형태와 유사하게 설계되었기 때문에 입력이 간편할 뿐만 아니라 입력과정중의 에러를 줄일 수 있고, 출력되는 결과값도 보기에 편리하다.

① 입력형태

만일 $ax+\frac{3\sin x}{4x}$ 와 같은 수학식이 있을 경우 컴퓨터에 이를 처리하기 위한 이제까지의 일반적인 방법은 (a*x)+(3*sin(x))/(4*x)와 같이 하였으나 Calc에서는 (ax)+(3sin(x))/(4x)나 ax+3sin x/4x와 같이 입력할 수도 있다.

물론 Calc는 이 세가지 방법이 모두 가능하며 Calc에 마련되어 있는 Compute 루틴을 이용하여 결과값을 계산하여 보면 모두 값이 나온다.

② 출력형태

Calc는 숫자 및 문자로 구성된 수학식을 입력받아 이를 미분하여 다시 숫자 및 문자로 구성된 문자열 형태로 된 수학식을 되돌려 준다. 예를 들어 3x+ax+sinx와 같은 식을 입력하면 미분된 3+a+cosx와 같은 문자열로 되돌려 준다.

③ 각도의 사용

Calc에서는 모든 값을 라디안으로 사용하나 sin 30'나 sin(x)'와 같이 단일 따옴표를 사용하여 각도를 그대로 입력할 수도 있으며 rad(x) 및 deg(x)가 준비되어 있어 서로간의 교환과 Compute루틴을 사용할 때 편리하다.

④ 곱셈기호의 생략

숫자계수와 문자계수 사이의 곱셈기호는 다음과 같이 생략할 수 있다.

5*a→5a, 5*sin(x)→5sin(x), 5*(3+4)→5(3+4)

또한 문자와 문자사이의 기호도 생략할 수 있다.

a*b*c→abc

연속된 괄호의 중간에서도 생략할 수 있다.

(...)*(...)→(...)(...)

⑤ 괄호의 생략

다음과 같이 맨 마지막의 우괄호를 생략할 수 있다.

5(sin(3x))→5(sin(3x

또한 함수명 다음에 단일항이 올때는 괄호를 생략할 수 있다.

sin(x)→sinx, sin(5x)→sin 5x, sin(5X^2)→sin 5X^2

⑥ 항 단위의 처리

Calc는 식을 항 단위로 처리한다. 그러므로 5X와 같은 식은 5*X와 5X의 두가지 입력형태가 가능한데 전자의 5*X는 2개의 항으로 5X는 하나의 항으로 취급한다. 또한 $\frac{4X^2}{8X^2}$ 와 같은 식도 (4*X)/(8X^2), (4X)/(8X^2), 4X/8X^2 등의 입력형태가 가능하고 Calc의 Compute루틴을 이용하여 결과값을 계산하여 보면 값이 나온다. 그러나 세번째의 식이

간단하고 항수도 적으므로 Diff를 이용하여 미분을 하여 보면 제일 간단한 형태의 출력이 나온다. 그러나 식을 정리하여 보면 같은 결과를 얻는다.

⑦ 식의 재조정

Calc는 다음과 같이 식을 재조정할 수 있는데 (3*(4))를 3*4로 3*1이나 3(1)^2에서 불필요한 괄호나 1을 제거하여 3으로 나타낸다. 또한 0*3+5의 0인 항을 제거하며 X^0나 3^0 등의 지수승이 0인 항을 1로 고쳐준다. 그리고 −3*(−4^3*(−5))와 같이 곱으로 연결된 항들의 부호를 계산하여 최종 부호를 첫번째항으로 옮겨준다.

(6) 자료구조

그러면 Calc가 수식을 어떻게 변형시켜 사용하는가 알아보자. 우선 이 부분에 관하여는 포인터(Pointer)형에 관하여 사전지식이 필요하며 단지 결과나 응용적인 측면에서만 관심이 있는 사람은 생략하여도 무관하다.

우선 가장 단순한 형태의 수식으로 Y=3+4+5를 살표보자. 이는 변수형의 선언에서

```
Type Polyhead = ^Polyheadtype;
    Polyheadtype = record
                    coef:real;
                    Ak:Polyhead;
                   end;
Var  Y:Polyhead;
```

와 같이 하고 이를 그림으로 표시하면 다음과 같다.

계수(coef)	연결 포인터(Ak)

예에 관하여 그려보면 그림 2의 (a)와 같다.

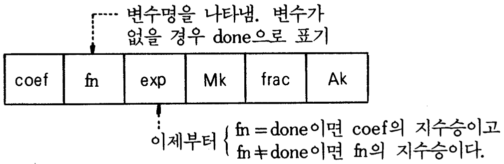
이제 지수승과 '*'으로 연결되는 항들을 나타내기 위한 새로운 필드인 exp와 Mk를 추가시켜 보자. Y=3+4+5일 경우는 다음과 같이 표현한다(그림 2의 (b) 참조).

다음은 '/'로 연결되는 항들을 표시하기 위하여 새로운 필드인 프랙션(Fraction)을 추가하여 보자. 여기서 프랙션은 '*'로 연결되었나 '/'로 연결되었나 하는 플래그의 기능을 나타냄을 알 수 있다. $Y=3^2+4/5$의 경우에는 전체적인 필드와 포인터의 구성은 다음과 같다(그림 2의 (c) 참조).

이제 수학식을 좀더 확장하여 변수를 도입한 경우를 생각해 보자. 이에 대한 Type 선언은 다음과 같다.

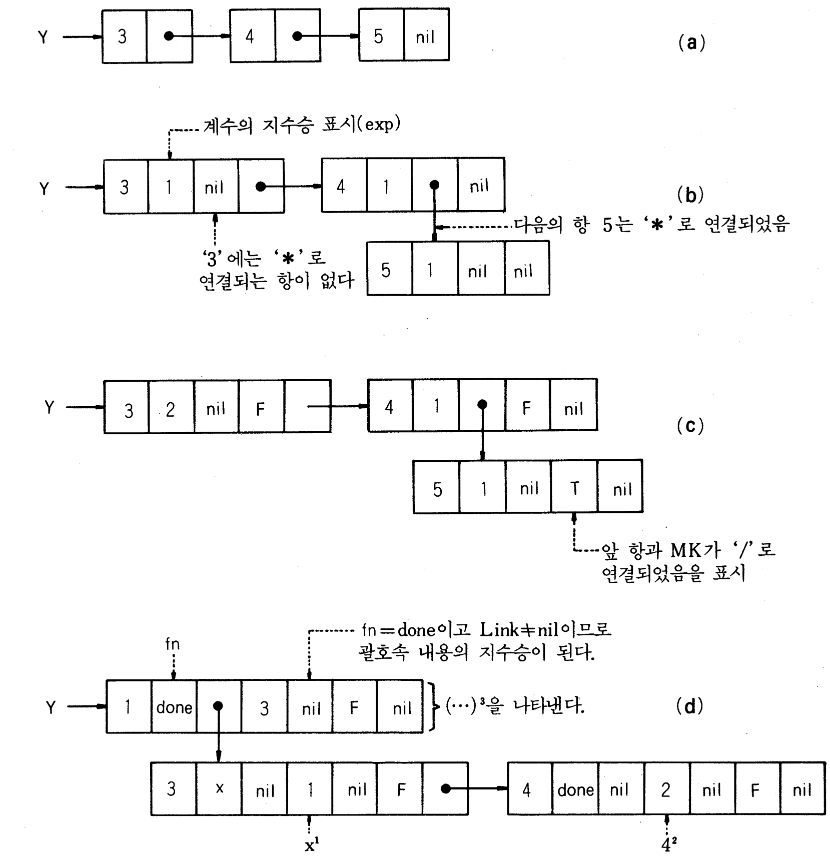
```
Type Polyhead = ^Polyheadtype;
    Ployheadtype = record
                    coef : real;
                    fn : string[5];
                    exp :real;
                    mk : Polyhead;
                    fraction : boolean;
                    Ak : Polyhead;
                  end;
```

자료의 구조를 나타내면 다음과 같다.



그리고 괄호가 있는 식에 관하여는 'Link'를 하나더 추가시킨 $Y=(3X+4^2)$은 다음과 같이 표현된다(그림 2의 (d)의

〔그림 2〕 자료구조의 표현



참조)

이제 전체적인 자료구조에 관하여 정리를 해보자.

coef	fn	link	exp	Mk	frac	Ak

coef : 계수
fn : 함수명
link : 함수의 내용이나 괄호속에 내용을 연결해 주는 연결 포인터
exp : 지수
Mk : 곱셈이나 나눗셈으로 연결된 항을 연결하는 포인터
fraction : 앞 항과의 관계가 나눗셈으로 연결되었는지의 여부를 나타냄
Ak : 덧셈이나 뺄셈으로 연결된 항을 연결하는 포인터

이외에도 여러가지 예를 들어 설명하고자 했지만 이해가 더 필요한 독자는 좀더 자세히 설명이 된 자료구조에 관한 서적을 참고 하길 바란다. 이러한 자료구조의 선언부분은 리스트 8의 Declare.inc에 나타나 있고 Make.inc가 실제 이 과정을 수행한다.

맺음말

이제까지 설명한 프로그램은 단지 미분에 관하여만 하였는데 이 프로그램은 공학도들이 필요한 여러가지 복잡한 계산을 처리해 줄 수 있는 곳에 응용될 수 있으므로 단순한 미분만이 아닌 여러가지 복잡한 수식의 처리를 간단하게 해결해 줄 수 있는 방법을 제시할 수 있을 것이다. 여기까지 부족한 글을 읽어준 분들께 감사를 드리며 이 프로그램을 좀 더 발전시켜 벡터나 적분에도 적용할 수 있는 프로그램이 나타나길 기대한다.

〔리스트 5〕

```
UNIT FUNC ;

interface

var errorcode : integer ;
    error     : boolean ;

procedure errorflag(code : integer) ;
FUNCTION   TAN(x : real) : real ;
FUNCTION   CSC(x : real) : real ;
FUNCTION   SEC(x : real) : real ;
FUNCTION   COT(x : real) : real ;
FUNCTION  ASIN(x : real) : real ;
FUNCTION  ACOS(x : real) : real ;
FUNCTION  ATAN(x : real) : real ;
FUNCTION  ACSC(x : real) : real ;
FUNCTION  ASEC(x : real) : real ;
FUNCTION  ACOT(x : real) : real ;
FUNCTION  SINH(x : real) : real ;
FUNCTION  COSH(x : real) : real ;
FUNCTION  TANH(x : real) : real ;
FUNCTION  CSCH(x : real) : real ;
FUNCTION  SECH(x : real) : real ;
FUNCTION  COTH(x : real) : real ;
FUNCTION ASINH(x : real) : real ;
FUNCTION ACOSH(x : real) : real ;
FUNCTION ATANH(x : real) : real ;
FUNCTION ACSCH(x : real) : real ;
FUNCTION ASECH(x : real) : real ;
FUNCTION ACOTH(x : real) : real ;
FUNCTION   LOG(x : real) : real ;
FUNCTION   RAD(x : real) : real ;
FUNCTION   DEG(x : real) : real ;
FUNCTION  RMOD(x, y : real) : real ;
FUNCTION  RDIV(x, y : real) : real ;
FUNCTION POWER(a, b : real) : real ;
FUNCTION FACTO(n    : real) : real ;
FUNCTION  PERM(a, b : real) : real ;
FUNCTION  COMB(a, b : real) : real ;

implementation

PROCEDURE ERRORFLAG(code : integer) ;
begin
  if code = 0 then error := false
              else error := true ;
  if (code = 0) or (errorcode = 0)
  then errorcode := code ;
end;


(*$I Func.inc *)


begin
  errorflag(0) ;
end.
```

〔리스트 6〕

```
FUNCTION TAN(x : real) : real ;
begin
  if cos(x) = 0 then errorflag( 1)
              else tan := sin(x) / cos(x) ;
end ;


FUNCTION CSC(x : real) : real ;
begin
  if sin(x) = 0 then errorflag( 2)
              else csc := 1.0 / sin(x) ;
end ;

FUNCTION SEC(x : real) : real ;
begin
  if cos(x) = 0 then errorflag( 3)
              else sec := 1.0 / cos(x) ;
end ;

FUNCTION COT(x : real) : real ;
begin
  if sin(x) = 0 then errorflag( 4)
              else cot := cos(x) / sin(x) ;
end ;

FUNCTION ASIN(x : real) : real ;
begin
  if abs(x) > 1.0
  then errorflag( 5)
  else if x = 1.0
       then asin := pi / 2
       else if x = -1.0
            then asin := -pi / 2
            else asin := arctan(x / sqrt(1.0 - x * x))
end ;

FUNCTION ACOS(x : real) : real ;
begin
  if abs(x) > 1.0 then errorflag( 6)
                else acos:= pi / 2 - asin(x)
end ;

FUNCTION ATAN(x : real) : real ;
begin
  atan := arctan(x) ;
end ;

FUNCTION ACSC(x : real) : real ;
begin
  if abs(x) < 1.0 then errorflag( 7)
                else acsc := asin(1.0 / x)
end ;

FUNCTION ASEC(x : real) : real ;
begin
  if abs(x) < 1.0 then errorflag( 8)
                else asec := pi / 2 - asin(1.0 / x)
end ;

FUNCTION ACOT(x : real) : real ;
begin
  acot := arctan(1.0 / x)
end ;

FUNCTION SINH(x : real) : real ;
begin
  sinh := (exp(x) - exp(-x)) / 2
end ;

FUNCTION COSH(x : real) : real ;
begin
  cosh := (exp(x) + exp(-x)) / 2
end ;

FUNCTION TANH(x : real) : real ;
begin
  tanh := sinh(x) / cosh(x)
end ;


FUNCTION CSCH(x : real) : real ;
begin
  if x = 0.0 then errorflag( 9)
           else csch := 1.0 / sinh(x)
end ;

FUNCTION SECH(x : real) : real ;
begin
  sech := 1.0 / csch(x)
end ;

FUNCTION COTH(x : real) : real ;
begin
  if x = 0.0 then errorflag(10)
           else coth := cosh(x) / sinh(x)
end ;


FUNCTION ASINH(x : real) : real ;
begin
  asinh := ln(x + sqrt(x * x + 1.0))
end ;

FUNCTION ACOSH(x : real) : real ;
begin
```

```
  if x < 1.0 then errorflag(11)
           else acosh := ln(x + sqrt(x * x - 1.0))
end ;

FUNCTION ATANH(x : real) : real ;
begin
  if abs(x) >= 1.0 then errorflag(12)
                  else atanh := ln((1.0 + x) / (1.0 - x)) / 2
end ;

FUNCTION ACSCH(x : real) : real ;
begin
  if x = 0.0 then errorflag(13)
           else acsch := asinh(1.0 / x)
end ;

FUNCTION ASECH(x : real) : real ;
begin
  if (x <= 0.0) or (x > 1.0) then errorflag(14)
                             else asech := acosh(1.0 / x)
end ;

FUNCTION ACOTH(x : real) : real ;
begin
  if abs(x) <= 1.0 then errorflag(15)
                  else acoth := atanh(1.0 / x)
end ;

FUNCTION LOG(x : real) : real ;
begin
  if x <= 0.0 then errorflag(16)
            else log := ln(x) / ln(10)
end ;

FUNCTION RAD(x : real) : real ;
begin
  rad := x * pi / 180
end ;

FUNCTION DEG(x : real) : real ;
begin
  deg := x * 180 / pi
end ;

FUNCTION RMOD(x, y : real) : real ;
begin
  if (int(x) <> x) or (int(y) <> y)
  then errorflag(17)
  else if int(y) = 0
       then errorflag(18)
       else rmod := trunc(x) mod trunc(y)
end ;

FUNCTION RDIV(x, y : real) : real ;
begin
  if (int(x) <> x) or (int(y) <> y)
  then errorflag(19)
  else if int(y) = 0
       then errorflag(20)
       else rdiv:= trunc(x) div trunc(y)
end ;

FUNCTION POWER(a, b : real) : real ;
var k : real ;
begin
  if a = 0.0
  then if b = 0.0
       then errorflag(21)
       else power := 0.0
  else begin
         if b = 1
         then power := a
         else if a < 0.0
              then if int(b) <> b
                   then errorflag(22)
                   else begin
                          k := exp(b * ln(abs(a))) ;
                          if odd(abs(trunc(b))) then k := -k ;
                          power := k ;
                        end
              else power := exp(b * ln(a)) ;
       end
end ;

FUNCTION FACTO(n : real) : real ;
var i : integer ;
    k : real ;
begin
  if int(n) <> n
  then errorflag(23)
  else if n < 0
       then errorflag(24)
       else begin
              k := 1.0 ;
              i := trunc(n) ;
              for i := i downto 1
              do k := k * i ;
              facto := k ;
            end
end ;

FUNCTION PERM(a, b : real) : real ;
var i, n : integer ;
    k : real ;
begin
```

```
if (int(a) <> a) or (int(b) <> b)
then errorflag(25)
else if (a < 0) or (b < 0)
     then errorflag(26)
     else if a < b
          then errorflag(27)
          else begin
                 i := trunc(a) ;
                 n := trunc(b) ;
                 k := 1.0 ;
                 while n > 0
                 do begin
                      k := k * i ;
                      i := i - 1 ;
                      n := n - 1 ;
                    end ;
                perm := k ;
              end ;
end ;

FUNCTION COMB(a, b : real) : real ;
var i, n : integer ;
    k : real ;
begin
  if (int(a) <> a) or (int(b) <> b)
  then errorflag(28)
  else if (a < 0.0) or (b < 0.0)
       then errorflag(29)
       else if a < b
            then errorflag(30)
            else begin
                   if (a-b) < b then b := a - b ;
                   comb := perm(a, b) / facto(b)
                 end ;
end ;
```

〔리스트 7〕

```
program CALC ;

(*$I  DECLARE.INC  *)
(*$I  UTIL.INC     *)
(*$I  MAKE.INC     *)
(* $I  SHOW.INC     *)
(*$I  ADJUST.INC   *)
(*$I  CONVERT.INC  *)
(*$I  COPYHD.INC   *)
(* $I  COMPUTE.INC *)
(*$I  DIFF.INC     *)

var ss,sg : string[255] ;
    polyY : polyhead ;
    kk    : real     ;
    n     : integer  ;

BEGIN
  init;
  repeat
    errorflag(0);
    writeln;  writeln;  writeln;
    for n := 1 to 79 do write('-'); writeln;
    writeln('TYPE  "/" --> Convert processflag ,  " * " -->  END.');
    writeln;  write('enter : ');
    readln(ss);
    writeln;
    if ss = '/'
    then begin
           processflag := not processflag;
           if processflag
           then writeln('...  Now Processflag ON   ...')
           else writeln('...  Now Processflag OFF  ...') ;
         end
    else if ss <> '*'
         then begin
                sg := diff(ss);
                writeln;
                writeln('Source : ', ss);
                writeln('Result : ', sg);
                if error then writeln(#7, 'ERROR : ', errorcode);
              end;
  until ss = '*';
end.
```

〔리스트 8〕

1

```
(*$V- *) <-- 모듈 Declare.inc
(*$C- *)
const VersionTitle = 'Version No 1.1   Mr. Jo ' ;
      sfn = 1  ;
      efn = 40 ;
      nul   = ''  ;
      blank = ' ' ;
      comma = ',' ;
      angle = #39 ;
      done  = 'DONE' ;
      funcs : array[sfn..efn] of
              string[5] = ('SIN'   , 'COS'   , 'TAN'  ,
                           'CSC'   , 'SEC'   , 'COT'  ,
                           'ASIN'  , 'ACOS'  , 'ATAN' ,
                           'ACSC'  , 'ASEC'  , 'ACOT' ,
                           'SINH'  , 'COSH'  , 'TANH' ,
                           'CSCH'  , 'SECH'  , 'COTH' ,
                           'ASINH' , 'ACOSH' , 'ATANH',
                           'ACSCH' , 'ASECH' , 'ACOTH',
                           'SQR'   , 'SQRT'  , 'EXP'  ,
                           'LN'    , 'LOG'   , 'RAD'  ,
                           'DEG'   , 'ABS'   , 'INT'  ,
                           'MOD'   , 'DIV'   , 'POWER',
                           '!'     , 'p'     , 'C'    ,
                           angle ) ;
TYPE
  S5   = string[ 5]  ;
  S10  = string[10]  ;
  S20  = string[20]  ;
  S255 = string[255] ;
  polyhead = ^polyheadtype ;
  polyheadtype = record
                  fn         : S5 ;
                  coef, exp  : real ;
                  fraction   : boolean ;
                  mk,ak,link : polyhead ;
                end ;
  polystack = ^polystacktype ;
  polystacktype = record
                   data : polyhead ;
                   link : polystack;
                   CC   : char     ;
                 end ;
VAR dgtbox : array ['a'..'z'] of real ;
    funcno : array [sfn..efn] of integer;
    stackdata   : polystack ;
    Hdstack     : polystack ;
    polydata    : polyhead  ;
    fncode      : integer ;
    TabConst    : integer ;
    errorcode   : integer ;
```

2

```
    error       : boolean ;
    processflag : boolean ;
    varname     : char    ;

PROCEDURE BELL ; <-- 모듈 Util.inc
begin
  write(#7)
end ;

FUNCTION LOWCASE(cc : char) : char ;
begin
  if cc in ['A'..'Z']
  then lowcase := chr(ord(cc) + ord(blank))
  else lowcase := cc
end ;

FUNCTION UPSTR(T : S20) : S20 ;
var i : integer ;
begin
  for i := 1 to length(T)
  do T[i] := upcase(T[i]) ;
  upstr := T
end ;

PROCEDURE SORTFUNCNAME ;
var i,j,k  : integer ;
    fn     : s5 ;
begin
  for i := sfn to efn
  do funcno[i] := i ;
  for i := sfn to efn
  do for j := (i) to efn
     do if funcs[i] > funcs[j]
        then begin
               fn := funcs[i] ; funcs[i] := funcs[j]  ; funcs[j]  := fn ;
               k := funcno[i] ; funcno[i]:= funcno[j] ; funcno[j] := k  ;
             end
end ;

FUNCTION FUNCTIONCODE(fn : s10 ) : integer ;
var first , last , mid : integer ;
begin
  if fn = angle
  then functioncode := 40
  else begin
         first  := sfn ;
         last   := efn ;
         functioncode := 0 ;
         while (first <= last) do
         begin
           mid := (first + last) div 2 ;
```

3

```
           if funcs[mid] = fn
           then begin
                  functioncode := funcno[mid] ;
                  first := last + 1 ;
                end
           else if funcs[mid] < fn
                then first := mid + 1
                else last  := mid - 1 ;
         end
       end
end ;

FUNCTION ISFUNCTION(fn : S10) : boolean ;
begin
  if functioncode(fn) <> 0
  then isfunction := true
  else isfunction := false
end ;

PROCEDURE ERRORFLAG(code : integer) ;
begin
  if code = 0
     then error := false
     else error := true ;
  if (code = 0) or (errorcode = 0)
     then errorcode := code
end ;

PROCEDURE INIT ;
var cc : char ;
begin
  sortfuncname   ;
  errorflag(0)   ;
  Tabconst := 2  ;
  varname := 'X' ;
  processflag := false ;
  for cc := 'a' to 'z' do dgtbox[cc] := 0 ;
  Hdstack^.link := nil
end ;

PROCEDURE PUSH(DD : polyhead ;  CC : char) ;
begin
  new(stackdata) ;
  stackdata^.data := DD ;
  stackdata^.CC   := CC ;
  stackdata^.link := Hdstack^.link ;
  Hdstack^.link   := stackdata
end ;

PROCEDURE POP(var DD : polyhead;  var CC : char) ;
begin
```

4

```
  if Hdstack^.link = nil
  then errorflag(51)
  else begin
         stackdata := Hdstack^.link ;
         Hdstack^.link := stackdata^.link ;
         DD := stackdata^.data ;
         CC := stackdata^.CC ;
         dispose(stackdata) ;
       end
end ;

PROCEDURE GETHD(var data : polyhead) ;
begin
  new(data) ;
  data^.coef := 1 ;
  data^.exp  := 1 ;
  data^.fn   := done ;
  data^.link := nil  ;
  data^.ak   := nil  ;
  data^.mk   := nil  ;
  data^.fraction := false ;
end ;

PROCEDURE INVESDATA(var y : polyhead) ;
var dd : polyhead ;
begin
  gethd(dd) ;
  dd^.coef := y^.coef ;
  dd^.fn   := y^.fn   ;
  dd^.link := y^.link ;
  dd^.exp  := y^.exp  ;
  y^.coef  := 1 ;
  y^.fn    := done ;
  y^.exp   := -1 ;
  y^.link  := dd ;
  y^.fraction := not y^.fraction ;
end ;

FUNCTION NOTOSTR(x : real) : S20 ;
var k    : integer ;
    U, T : S20     ;
begin
  str(x:0:TabConst , U) ;
  if TabConst <> 0
  then while U[length(U)] = '0'
       do delete(U,length(U),1) ;
  k := 0 ;
  if U[length(U)] = '.'
  then begin
         delete(U,length(U),1) ;
```

5

```
         while length(U) > 11 do
         begin
           k := k + 1 ;
           delete(U,length(U),1) ;
         end
       end ;
  if k > 0 then begin
                  str(k,T) ;
                  U := U + 'E+' + T ;
                end ;
  if U = '-0' then U := '0' ;
  notostr := U  ;
end ;

FUNCTION LASTCHR(T : S255) : char ;
begin
  T :='#' + T ;
  while T[length(T)] = ' '
  do delete(T,length(T),1) ;
  if T <> '#' then lastchr := T[length(T)]
              else lastchr := ' ' ;
  delete(T,1,1) ;
end ;

PROCEDURE ERASEHD(var d : polyhead) ;
var memo : polystack;
    cc   : char     ;
begin
  memo := hdstack^.link ;
  if d <> nil then push(d,cc) ;
  while hdstack^.link <> memo do
  begin
    pop(d,cc) ;
    if d^.link <> nil then push(d^.link,cc) ;
    if d^.mk <> nil then push(d^.mk,cc) ;
    if d^.ak <> nil then push(d^.ak,cc) ;
    dispose(d)
  end
end ;

FUNCTION DEVIDE(a, b : real) : real ;
begin
  if b = 0 then errorflag(52)
           else devide := a / b
end ;

FUNCTION VALUE(T : S20; var err : boolean) : real ;
var status : integer;
    x      : real    ;
begin
```

6

```
  val(T, x, status) ;
  if status = 0
  then begin  value := x   ;  err := false ;   end
  else begin  value := 0.0 ;  err := true  ;   end ;
end ;

FUNCTION ISDGT(T : S20) : boolean ;
var status : integer ;
    No     : real ;
begin
  val(T,No,status) ;
  if status = 0 then isdgt := true
                else isdgt := false
end ;

FUNCTION POSTOPERATOR(T : S10) : boolean ;
begin
  if (  (length(T) = 1) and (T[1] in ['!','P','C',angle]) )
     or (T = 'MOD')
     or (T = 'DIV')
  then postoperator := true
  else postoperator := false
end ;

FUNCTION MakeHd(S : S255) : polyhead ; <-- 모듈 Make.inc
label errorexit ;
var T      : S255 ;
    U      : S20  ;
    cc     : char ;
    No     : real ;
    n      : integer  ;
    err    : boolean  ;
    MemHd  : polyhead ;
    Hd     : polyhead ;
    Memo   : polyStack;

FUNCTION gettoken(var S : S255) : S20 ;
var U : S20 ;
    i : integer ;
begin
  U := nul ;
  i := 1   ;
  while (S[1] in [comma, '#']) do delete(S,1,1) ;
  while (S[i] <> comma) and (i < length(s)) do
  begin
    U := U + S[i] ;
    i := i + 1 ;
  end ;
  delete(S,1,i) ;
  gettoken := U ;
```

7

```
end ;

FUNCTION nexttoken(S : S255) : S20 ;
begin
  nexttoken := gettoken(S) ;
end ;

PROCEDURE SEPARATE(var S : S255) ;
label errorexit ;
var chrset, dgtset, opset : set of char ;
    i, lpno, rpno : integer ;
    pass   : boolean ;
    U      : S20 ;
begin
 chrset := ['a'..'z' , 'A'..'Z'] ;
 dgtset := ['0'..'9' , '.', 'E'] ;
 opset  := ['*','/','+','-','^','C','P'] ;
 T   := nul ;
 lpno := 0  ;
 rpno := 0  ;
 for i := 1 to length(S)
 do if S[i] = blank
    then begin
           if (S[i+1] in chrset) and (T[length(T)] in chrset)
           then T := T + '1'
         end
    else begin
           T := T + S[i] ;
           if S[i] = '('
           then lpno := lpno + 1
           else if S[i] = ')'
                then rpno := rpno + 1
         end ;

 if T = nul
 then T := '0'
 else if T[1] = '+'
      then delete(T,1,1)
      else if T[1] in [')','*','/','^','!','"','C','P',angle]
           then begin
                  errorflag(61) ;
                  goto errorexit;
                end
           else if T[length(T)] in opset
                then begin
                       errorflag(62) ;
                       goto errorexit;
                     end ;
 if (lpno < rpno)
 then begin
        errorflag(63) ;
```

8

```
        goto errorexit;
      end
 else while lpno <> rpno
      do begin
           T    := T + ')' ;
           rpno := rpno + 1;
         end ;
 i := 1 ;
 S := nul ;
 T := T + '#' ;
 while T[i] <> '#' do
 begin
  pass := false ;
  case  T[i] of
   '(' : begin
           if T[i+1] = '+'
           then delete(T,i+1,1)
           else if T[i+1] in ['*','/','^','!','"','C','P',angle]
                then begin
                       errorflag(64) ;
                       goto errorexit;
                     end ;
           S := S + T[i] ;
           i := i + 1 ;
         end ;
   '-','+',')','*','/','^' :
         begin
           if (T[i] = ')')
           then begin
                  if (T[i-1] in opset) or (T[i-1] = '(')
                  then begin
                         errorflag(65) ;
                         goto errorexit;
                       end
                end
           else if (T[i] in opset) and (T[i+1] in opset)
                then begin
                       errorflag(66) ;
                       goto errorexit;
                     end ;
           if T[i] = '-'
           then if (i = 1) or (T[i-1] = '(')
                then if (T[i+1] in ['0'..'9'])
                     then begin
                            S := S + '-' ;  (* -2 or ...(-3...  *)
                            pass := true ;
                          end
                     else S := S + '-1'          (* '-X' -> '-1 * X'  *)
                else S := S + '+,-1'             (* 2-3  -> 2+(-1)*3  *)
           else S := S + T[i] ;
           if (T[i] = ')') and (T[i+1] = '(')  (* (..)(..) -> (..)*(..) *)
```

9

```
           then S := S + comma + '*' ;
           i := i + 1 ;
          end ;
  '0'..'9' :
         repeat
           S := S + T[i] ;
           if(T[i] = 'E') and (T[i+1] in ['+','-'])
           then begin
                  i := i + 1 ;
                  S := S + T[i] ;
                end ;
           i := i + 1 ;
         until not(T[i] in dgtset) ;
  'a'..'z' : begin
               U := nul ;
               repeat
                 U := U + T[i] ;
                 i := i + 1    ;
               until not (T[i] in ['a'..'z']) ;
               if isfunction(upstr(U))
               then if (U <> 'c') and (U <> 'p')
                    then U := upstr(U) ;
               S := S + U ;
             end ;
  'A'..'Z' : repeat
               S := S + T[i] ;
               i := i + 1    ;
             until not (T[i] in ['A'..'Z']) ;
   '"','!',
   angle : begin
             if (T[i] = '"') then T[i] := angle ;
             S := S + T[i] ;
             i := i + 1 ;
           end ;
   else    (* case else *)
   begin
     if T[1] <> '#' then errorflag(67) ;
     goto errorexit ;
   end
  end ;
   if not pass then S := S + comma ;
  end ;
 errorexit:
end ;

begin
 separate(s) ;
 Memo := HdStack^.link ;
 getHd(Hd) ;
 push(Hd,' ') ;
 MemHd := Hd  ;
```

10

```
if postoperator(nexttoken(S))
then errorflag(68) ;
while (not error) and (S <> nul) do
begin
  T := gettoken(S) ;
if (isfunction(T))
then begin
       U := nexttoken(S) ;
       if not( (T = '!') or (T = angle) ) and ( U = '' )
       then errorflag(69) ;
     end ;
if (isfunction(T)) or (T[1] in ['a'..'z',angle]) or (T = varname)
then begin
       if T[1] in ['a'..'z']
       then begin
              U := T ;
              T := T[1] ;
              delete(U,1,1) ;
              if length(U) <> 0
              then S := '*,' + U + comma + S     (* ab --> a * b   *)
              else begin
                     U := nexttoken(S) ;
                     if (length(U) = 1) and (U[1] = varname)
                     then S := '*,' + S ;        (*   aX -> a * X   *)
                   end ;
              if (upcase(T) = varname) then T := upcase(T) ;
            end ;
       if Hd^.fn <> done
       then errorflag(70)
       else if Hd^.exp <> 1
            then errorflag(71)
            else if (Hd^.link = nil) or ( postoperator(T) )
                 then begin
                        U := nexttoken(S) ;
                        if (not (T[1] in ['a'..'z','!',angle,varname]))
                           and (U[1] in ['*','/','+','-','^'])
                        then errorflag(72)
                        else Hd^.fn := T ;
                      end
                 else errorflag(73) ;
       if error then goto errorexit ;
       if isfunction(T) and (T <> varname) and (T <> angle) and (T <>'!'
       then if (T = 'P') or (T = 'C') or (T = 'MOD') or (T = 'DIV')
            then S := '^,' + S
            else begin
                   push(Hd,'F') ;
                   getHd(polydata) ;
                   Hd^.link := polydata ;
                   Hd := Hd^.link ;
                   push(Hd,'F') ;
                 end
```

11

```
     end
else
begin
  case T[1] of
   '+' : begin
           pop(Hd,CC) ;
           while CC in ['*' ,'/', 'F']
           do pop(Hd,CC) ;
           getHd(polydata) ;
           Hd^.ak := polydata ;
           Hd := Hd^.ak ;
           push(Hd,'+') ;
         end ;
   '*' ,
   '/' : begin
           pop(polydata,CC) ;
           if CC in ['F'] then pop(Hd,CC) ;
           push(polydata,CC) ;
           getHd(polydata)   ;
           while Hd^.mk <> nil
           do Hd := Hd^.mk ;
           Hd^.mk := polydata ;
           Hd := Hd^.mk ;
           if T = '/' then Hd^.fraction := true ;
         end ;
   '(' : if postoperator(nexttoken(S))
         then errorflag(74)
         else begin
                push(Hd,'(') ;
                getHd(polydata) ;
                Hd^.link := polydata ;
                Hd := Hd^.link ;
                push(Hd,' ')   ;
              end ;
   ')' : repeat
           pop(Hd,CC) ;
         until (CC = '(') and (not error) ;
  '0'..'9',
   '-' : if Hd^.fn <> done then errorflag(75)
         else if (Hd^.link <> nil) then errorflag(76)
         else begin
                No := value(T,err) ;
                if err then errorflag(77) ;
                if (Hd^.coef =  1)
                then Hd^.coef := No
                else if (Hd^.coef = -1) and (nexttoken(S) <> '^')
                     then Hd^.coef := -1 * No   (* 2-3 => 2,+,-1,3 *)
                     else begin                 (*    = 2 + (-3)  *)
                            getHd(polydata) ;(* 2-3^2 -> 2-(3)^2  *)
                            polydata^.coef := No ;(* (2)(3)(4)-> 2(3)*4 *)
                            Hd^.link := polydata ;
```

12

```
                      if isdgt(nexttoken(S)) then S := '*,' + S ;
                    end
            end ;
    '^' : if (Hd^.fn = angle) or (Hd^.fn = '!')
          then errorflag(78)
          else if Hd^.exp = 1
               then begin
                      T := gettoken(S) ;
                      if (T[1] = '(')
                      then begin
                             T := gettoken(S) ;
                             U := gettoken(S) ;
                             if (U[1] <> ')')
                             then errorflag(79)
                           end ;
                      if not error
                      then begin
                             Hd^.exp := value(T,err) ;
                             if err then errorflag(80) ;
                           end
                    end
               else errorflag(81) ;
    else errorflag(82)
   end  (* case else *)
  end   (* while if  *)
 end ;    (* while *)
errorexit:
 while HdStack^.link <> Memo
 do pop(polydata,CC) ;
 makehd := MemHd ;
end ;

PROCEDURE DELPRHD(var y : head) ; <-- 모듈 Adjust.inc
var U     : S4 ;
    again : boolean ;

  procedure delprhead(var y : head;  var again : boolean) ;
  begin
    if (y^.exp = 1)
    then begin
           ddd    := y^.mk   ;
           y^.mk  := dd^.mk ;
           dd     := y      ;
           while dd^.mk <> nil do dd := dd^.mk ;
           dd^.mk := ddd  ;
         end ;
    dd := y^.link ;
    y^.fraction := dd^.fraction  ;
    y^.coef:= y^.coef * dd^.coef ;
    y^.exp := y^.exp  * dd^.exp  ;
    y^.fn  := dd^.fn  ;
```

13

```
   ddd    := dd^.link;
   dispose(y^.link) ;
   y^.link := ddd    ;
   again := true     ;
  end ;

begin
  repeat
   again := false  ;
   dd    := y^.link;
   if (y^.fraction = false) and (y^.fn = done) and (dd <> nil)
   then if (dd^.ak = nil)
        then if (y^.exp = 1)
             then begin
                    if (y^.coef = 1)
                    then delprhead(y,again)
                    else if (dd^.coef = 1) and
                            (not ((dd^.fn = done) and (dd^.link <> nil)))
                         then delprhead(y,again)
                  end
             else if (dd^.mk = nil)
                  then begin
                         U := dd^.fn  ;
                         if (y^.coef = 1)
                           and (dd^.fn = done)
                           and (dd^.link = nil)
                         then delprhead(y,again)
                         else if (dd^.coef = 1)
                              then if ((dd^.fn = done) and (dd^.link <> nil))
                                      or (length(U) = 1)
                                         (* U[1] in ['a'..'z', X]  *)
                                   then delprhead(y,again)
                       end ;
  until not again ;
end ;


PROCEDURE MOVESIGN(var d, y : head) ;
var moveflag : boolean  ;
begin
  if (d <> y)
  then
begin
  if    (y^.fn       = done  )
    and (y^.link     = nil   )
    and (int(y^.exp) = y^.exp)
    and (not odd(trunc(y^.exp)))
  then begin
         y^.coef := -y^.coef ;
         moveflag := false   ;
       end
```

14

```
   else if (int(y^.exp) <> y^.exp)
           and (y^.fn   = done   )
           and (y^.link = nil    )
        then moveflag := false
        else moveflag := true ;
   if    moveflag
     and (d^.fn   = done)
     and (d^.link = nil )
     and (d^.exp  <> 1  )
   then begin
          gethd(data) ;
          data^.coef := d^.coef ;
          d^.coef := 1 ;
          d^.link := data ;
        end ;
   if moveflag
   then begin
          y^.coef := - y^.coef ;
          d^.coef := - d^.coef ;
        end ;
  end ;
end ;


PROCEDURE CONVERTFRACTION(var y : head) ;
begin
  if   ((y^.fn = done) and (y^.link = nil))
    or ((y^.fn = X    ) and (y^.coef = 1)  )
  then begin
         y^.fraction := not y^.fraction ;
         y^.exp := - y^.exp ;
       end ;
end ;


PROCEDURE ZEROEXP(var y : head) ;
begin
  if y^.fn = X
  then begin          (* 6x^0  --> 6 *)
         y^.fn  := done ;
         y^.exp := 1 ;
       end
  else if y^.link = nil
       then begin
              y^.coef := 1 ;
              y^.exp  := 1 ;
            end ;
end ;


PROCEDURE MKONETEST(var y : head) ;
```

15

```
begin
  dd := y^.mk ;
  if     (dd^.coef = 1   )
     and (dd^.fn   = done)
     and (dd^.link = nil )
     and (dd^.ak   = nil )
  then begin
         ddd := dd^.mk ;
         dispose(y^.mk);
         y^.mk := ddd   ;
       end ;
end ;


PROCEDURE LINKONETEST(var y : head) ;
begin
  dd := y^.link   ;
  if (y^.fn = done)
     and (dd^.coef = 1   )
     and (dd^.fn   = done)
     and (dd^.link = nil )
     and (dd^.ak   = nil )
     and (dd^.mk   = nil )
  then begin
         dispose(y^.link) ;
         y^.link := nil    ;
         y^.exp  := 1      ;          (*  ?(1)^?   --> ? *)
       end ;
end ;


PROCEDURE ADJUST(var y : head) ;
var flag, delprflag, again : boolean ;
    stackMemo, Memo : Stack ;
    c , cc          : char  ;
begin
  stackmemo := Hdstack^.link ;
  f := y  ;
  push(f,blank) ;
  while (hdstack^.link <> stackmemo)
  do begin                          (* while - 1 - *)
       pop(f,cc) ;
       if (cc <> '#')
       then
       begin
         if (cc = 'L')
         then begin
                if isfunc(f^.fn)
                then delprflag := false
                else delprflag := true ;
                hd := f ;
```

16

```
            f  := f^.link ;
          end
      else delprflag := true ;
      memo := hdstack^.link  ;
      while (f <> nil)
      do begin                    (* while - 2 - *)
           d := f;   f := f^.ak;   d^.ak := nil;    y := d;
           flag := false ;
           while (y <> nil) and (not flag)
           do begin               (* while - 3 - *)
                if (y^.fn = done) and (delprflag) then delprhd(y) ;
                if (y^.coef = 0 )
                then flag := true
                else if (y^.fn = done) and (y^.link <> nil)
                     then if (y^.link^.coef = 0) and (y^.link^.ak = nil)
                          then flag := true ;
                if (y^.coef < 0 ) then movesign(d ,y) ;
                if (y^.exp  < 0 ) then convertfraction(y) ;
                if (y^.exp  = 0 ) then zeroexp(y) ;
                if (y^.mk   <> nil) then mkonetest(y) ;
                if (y^.link <> nil) then linkonetest(y) ;
                y := y^.mk ;
              end ;               (* end while   - 3 - *)
           if flag then erasehd(d)
                   else push(d, blank) ;
         end ;                    (* end while   - 2 - *)
       if (hdstack^.link = memo)
       then begin
              gethd(f) ;
              f^.coef := 0 ;
            end
       else begin
              pop(f, c) ;
              while (hdstack^.link <> memo)
              do begin
                   pop(d, c) ;
                   d^.ak := f;
                   f := d ;
                 end ;
            end ;
       if cc = 'L' then hd^.link := f ;
       while (f <> nil)
       do begin
            push(f,'#') ;
            if f^.mk <> nil then push(f^.mk, 'M') ;
            if f^.link <> nil then push(f, 'L') ;
            f := f^.ak ;
          end ;
     end ;     (* if cc = # *)
   end ;       (* end while   - 1 - *)
 while (Hdstack^.link <> StackMemo)
```

17

```
   do pop(d,c) ;
   y := f ;
 end ;      (* procedure adjust *)

 UNCTION CONVERT( Y : polyhead) : S255 ; <-- 모듈 Convert.inc
 const nul = '' ;
 VAR fractionflag  : boolean;
     T, TS  : S255 ;
     U      : S5   ;
     CC     : char ;
     i , j  : integer;
     D      : polyHead   ;
     Memo   : polyStack  ;
 begin
   Memo := HdStack^.link ;
   T    := nul ;
   if y = nil
   then T := '0'
   else if not error
        then begin
               if Y^.fraction
               then fractionflag := true
               else fractionflag := false;
               push(Y,'0');
             end;
   while HdStack^.link <> Memo do
   begin
     pop(Y,CC);
     case CC of
      '0':begin
            push(Y,'1');
            if ((Y^.fn = done) and (Y^.link = nil)) or (abs(Y^.coef) <> 1)
            then begin
                   TS := notostr(Y^.coef);
                   if (Y^.coef < 0)
                   then if lastchr(T) in ['*' , '/']
                        then TS := '(' + TS + ')'
                        else if (lastchr(T) in ['+' , '-'])
                             then if (int(Y^.exp) = Y^.exp)
  (* 2+(-4)^2 -> 2+(-4)^2 *)           and (not odd(trunc(Y^.exp)))
  (* 2+(-4)^3 -> 2 - 4 ^3 *)           and (Y^.fn   = done)
                                       and (Y^.link = nil)
                                  then begin
                                         delete(T,length(T),1);
                                         T  := T + '+' ;
                                         TS := '(' + TS + ')' ;
                                       end
                                  else delete(TS,1,1);
                   T := T + ' ' + TS ;
                 end
            else if y^.coef = -1
```

18

```
            then if lastchr(T) in ['*','/']
                 then t := t + ' (-1)'
                 else if (lastchr(T) <> '-')
                      then t := t + '-' ;
       U := y^.fn ;
       if (Y^.link <> nil) and ( (U = done) or postoperator(U) )
       then begin
              T := T + ' (';
              push(Y^.link,'0')
            end
       else if (U <> done)
            then begin
                   if (U[1] in ['a'..'z', varname,'!',angle])
                     and (lastchr(T) in ['0'..'9'])
                   then T := T + Y^.fn
                   else T := T + ' ' + Y^.fn;
                   if Y^.link <> nil then push(Y^.link,'0');
                 end ;
        if Y^.link <> nil
        then if Y^.link^.fraction
             then fractionflag := true ;
     end;
  '1':begin
        push(Y,'2');
        U := Y^.fn ;
        if (Y^.link <> nil)
        then begin
               if (Y^.fn = done)
               then T := T + ')'
               else if postoperator(U)
                    then T := T + ')' + U ;
             end;
        if (Y^.exp <> 1)
           or (fractionflag)
           or (U = 'P'  )
           or (U = 'C'  )
           or (U = 'DIV')
           or (U = 'MOD')
        then begin
               if fractionflag
                 then TS := notostr(-Y^.exp)
                 else TS := notostr(Y^.exp);
               if TS[1] = '-' then TS := '(' + TS + ')';
               if    (U = 'P'  )
                  or (U = 'C'  )
                  or (U = 'DIV')
                  or (U = 'MOD')
               then T := T + ' ' + TS
               else T := T + '^' + TS ;
             end;
        if Y^.MK <> nil
```

19

```
        then begin
               if Y^.mk^.fraction then T := T + ' /'
                                   else T := T + ' *';
               push(Y^.MK,'0');
             end ;
          fractionflag := false ;
        end;
    '2':begin
          push(Y,'3');
          if Y^.ak <> nil
          then begin
               if Y^.ak^.coef < 0 then T := T + ' -'
                                   else T := T + ' +';
               push(Y^.ak,'0') ;
               if Y^.ak^.fraction then fractionflag := true ;
             end
        end
  end
end;
while HdStack^.link <> Memo
do begin
     pop(d,cc);
     dispose(d);
   end;
if copy(T,1,2) = '- ' then delete(T,2,1);
convert := T;

FUNCTION COPYHD(Y:polyHead):polyHead; <-- 모듈 Copyhd.inc
VAR Memory,DD: polyHead;
    Memo    : polyStack;
    CC      : char;
begin
  if Y = nil then CopyHd := nil
  else
  begin
    Memo := HdStack^.link ;
    push(Y,'0');
    GetHd(polydata);
    push(polydata,'0');
    Memory := polydata;
    while HdStack^.link <> Memo do
    begin
      pop(polydata,CC);
      pop(Y,CC);
      case CC of
      '0' : while Y <> nil do
            begin
              polydata^.fraction := Y^.fraction;
              polydata^.coef := Y^.coef;
              polydata^.exp := Y^.exp;
              polydata^.fn := Y^.fn;
```

20

```
                push(Y,'1');
                push(polydata,CC);
                if Y^.link <> nil then
                begin
                  GetHd(DD);
                  polydata^.link := DD;
                  polydata := polydata^.link;
                end;
                Y := Y^.link;
              end;
        '1':begin
              push(Y,'2');
              push(polydata,CC);
              if Y^.mk <> nil then
              begin
                GetHd(DD);
                polydata^.mk := DD;
                push(Y^.mk,'0');
                push(polydata^.mk,CC);
              end;
            end;
        '2':begin
              push(Y,'3');
              push(polydata,CC);
              if Y^.ak <> nil then
              begin
                GetHd(DD);
                polydata^.ak := DD;
                push(Y^.ak,'0');
                push(polydata^.ak,CC);
              end;
            end;
        end;  (* case *)
      end;    (* WHILE *)
    CopyHd := Memory  ;
    end;    (* if *)
  end;
end;

 procedure middle(y,d,memory:polyhead);<-- 모듈 Diff.inc
var ccc : char ;
    kk,yy : polyhead ;
    sa : s255 ;
begin
  writeln ;
  writeln ;  kk := y^.ak ; y^.ak := nil;
  sa := convert(y);  y^.ak := kk ;  writeln('Y : ',sa);
  if d <> nil
  then begin
         yy := copyhd(d) ;
         adjust(yy) ;
```

21

```
         sa := convert(yy);
         writeln('D : ',sa);
         erasehd(yy) ;
       end;
  yy := copyhd(memory) ;
  adjust(yy) ;
  sa := convert(yy);
  writeln;
  writeln('Diffrential : ',sa);
  write('Any key to continue.....') ;
  erasehd(yy) ;
  readln(kbd,ccc);
end;

FUNCTION DIFFHD( y : polyhead) : polyhead;
var U : string[5] ;
    c, cc     : char ;
    stackmemo : polystack;
    d,dd, memory : polyhead ;

PROCEDURE FUNCPROCESS(var d : polyhead; y : polyhead);
var C      : S5  ;
    fncode  : integer ;
    dd, ddd : polyhead;
begin
  d^.coef := y^.coef;
  C :=  y^.fn;
  fncode := functioncode(y^.fn) ;
  if (fncode in [2, 4, 6, 8, 10, 12, 16, 17, 18, 22, 23])
     then d^.coef := -d^.coef ;
  case  fncode  of
    1,2,13,14 : begin  (* SIN *)
                  case fncode of
                    1  : d^.fn    := 'COS'  ;
                    2  : d^.fn    := 'SIN'  ;
                    13 : d^.fn    := 'COSH' ;
                    14 : d^.fn    := 'SINH' ;
                  end ;
                  d^.link := copyhd(y^.link)
                end ;
    3,6,15,18 : begin  (* TAN *)
                  d^.exp := 2;
                  gethd(dd);
                  d^.link := dd;
                  case fncode of
                    3  : dd^.fn := 'SEC' ;
                    6  : dd^.fn := 'CSC' ;
                    15 : dd^.fn := 'SECH' ;
                    18 : dd^.fn := 'CSCH' ;
                  end;
                    dd^.link:= copyhd(y^.link);
```

22

```
          end ;
4,5,16,17 : begin  (* SEC *)
             gethd(dd);
             d^.fn := y^.fn ;
             d^.link :=  copyhd(y^.link);
             case fncode of
               4  : dd^.fn := 'COT' ;
               5  : dd^.fn := 'TAN' ;
               16 : dd^.fn := 'COTH' ;
               17 : dd^.fn := 'TANH' ;
             end;
             dd^.link:=  copyhd(y^.link);
             pop(d,cc);
             d^.mk := dd ;
             d := d^.mk ;
             push(d,'0');
          end ;
7..12 ,
19..24 : begin
            gethd(dd);
            d^.mk := dd;
            dd^.fraction := true ;
            if (fncode in [10, 11, 22, 23])
            then begin
                   gethd(ddd) ;
                   dd^.link := ddd ;
                   dd  := dd^.link ;
                   dd^.link := copyhd(y^.link) ;
                   gethd(ddd) ;
                   dd^.mk := ddd ;
                   dd := dd^.mk ;
                 end;
            if (fncode in [7, 8, 10, 11, 19, 20, 22,23])
            then begin
                   dd^.fn := 'SQRT' ;
                   gethd(ddd);
                   dd^.link := ddd ;
                   dd := dd^.link ;
                 end;
            gethd(ddd);
            dd^.link := ddd ;
            dd := dd^.link ;
            gethd(ddd);
            dd^.ak   := ddd;
            if (fncode in [7, 8, 10, 11, 20, 21, 22, 23])
            then ddd^.coef := -ddd^.coef ;
            if (fncode in [10, 11, 19, 20])
            then ddd := dd ;
            ddd^.exp := 2 ;
            ddd^.link:= copyhd(y^.link);
          end ;   (*  if  'ARC' 7..12 , 19,24  *)
```

23

```
    25 : begin   (* SQR *)
           d^.coef := d^.coef * 2.0 ;
           d^.link := copyhd(y^.link);
         end ;
    26 : begin  (* SQRT *)
           gethd(dd);
           d^.mk := dd;
           dd^.coef := 2 ;
           dd^.fn := 'SQRT';
           dd^.fraction := true ;
           dd^.link := copyhd(y^.link);
         end ;
    27 : begin  (* EXP *)
           d^.fn   := 'EXP';
           d^.link := copyhd(y^.link);
         end ;
    28 : begin   (* LN *)
           d^.exp  := -1 ;
           d^.link := copyhd(y^.link);
         end ;
    else errorflag(91) ;
  end ;  (* case *)
1;  (* procedure *)

gin
stackmemo := hdstack^.link ;
push(y,'0') ;
gethd(d)    ;
memory := d ;
push(d,'0') ;
while (not error) and (hdstack^.link <> stackmemo) do
begin
  pop(d,c);
  pop(y,cc);
  case cc of
   '0' : begin
           if y^.fraction then invesdata(y);
           push(y,'1');
           push(d,'1');
           U := y^.fn ;
           if (y^.fn = done) or (U[1] in ['a'..'z'])
           then
           begin
             if (y^.link = nil) or (y^.exp = 0)
             then d^.coef := 0
             else begin
                    if y^.exp = 1
                    then d^.coef := y^.coef
                    else begin
                           d^.coef := y^.coef * y^.exp;
                           d^.exp  := y^.exp - 1;
```

24

```
                      d^.fn   := y^.fn;
                      d^.link := copyhd(y^.link);
                      gethd(dd);
                      d^.mk := dd;
                      d := d^.mk;
                    end;
                 gethd(dd);
                 d^.link := dd;
                push(y^.link,'0');
                push(d^.link,'0');
               end
         end
         else if y^.fn = varname
              then begin
                     d^.coef := y^.coef * y^.exp;
                     d^.exp  := y^.exp - 1;
                     d^.fn   := y^.fn;
                   end
         else begin
                funcprocess(d,y);
                gethd(dd);
                while d^.mk <> nil
                do d := d^.mk;
                d^.mk := dd ;
                d := d^.mk;
                push(y^.link,'0');
                push(d,'0');
              end;
       end;(* case '0' *)
  '1' : begin
         push(y,'2');
         push(d,'2');
         if y^.mk <> nil
         then begin
                while d^.mk <> nil
                do d := d^.mk;
                d^.mk := copyhd(y^.mk);
                pop(d,c);
                gethd(dd);
                d^.ak   := dd     ;
                d       := d^.ak   ;
                d^.coef:= y^.coef;
                d^.exp := y^.exp ;
                d^.fn  := y^.fn   ;
                d^.link:= copyhd(y^.link);
                push(d,'2');
                gethd(dd)   ;
                d^.mk := dd;
                d := d^.mk ;
                if y^.mk^.mk <> nil
                then begin
```

```
                        gethd(dd)    ;
                        d^.link := dd;
                        d := d^.link ;
                      end;
                  push(y^.mk,'0');
                  push(d,'0');
                end;
           end;
      '2' : begin
             push(y,'3');
             push(d,'3');
             if processflag then middle(y,d,memory) ;
             if y^.ak <> nil
             then begin
                    gethd(dd);
                    d^.ak := dd;
                    push(y^.ak,'0');
                    push(d^.ak,'0');
                  end;
           end;
      end;   (* case *)
 end;    (* while *)
 while (hdstack^.link <> stackmemo)
```

```
   do pop(d,c) ;
   diffhd := memory;
end;   (* diff - HD *)

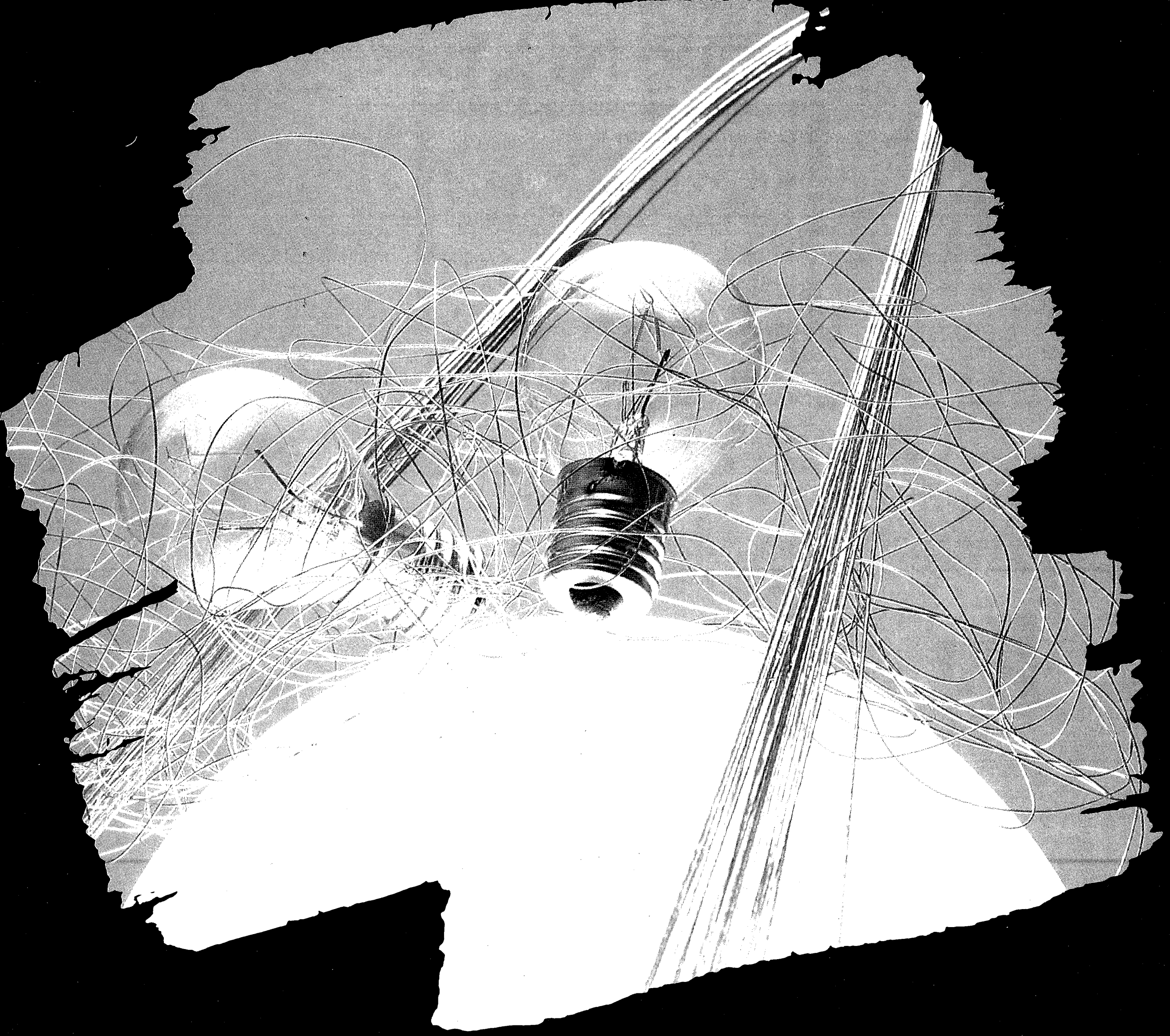
FUNCTION DIFF(ss : S255 ) : S255 ;
var polyY,polyD : polyhead ;
begin
  if error
  then diff := 'BEFORE ERROR'
  else begin
         polyY := makehd(ss) ;
         if error
         then diff :=  'MAKE ERRIR'
         else begin
                adjust(polyY); adjust(polyY);
                polyD := diffhd(polyY);
                adsust(polyD); adjust(polyD);
                diff := convert(polyD);
                erasehd(polyD);
              end;
         erasehd(polyY);
       end;
end;
```

소중한 당신의 지식을 나누어 가지고자 합니다

당신에게는 아주 사소한 것일지라도 저희는 귀중하게 여깁니다.
혼신의 힘을 기울여 얻어낸 대작보다도 어쩌면 졸작이 더 귀할 수도 있읍니다.
아무런 주저마시고 저희 지면을 이용하십시오.
저희는 당신의 지식을 귀히 여기겠읍니다.
지면 이용방법은
먼저 (02)754—1044로 전화를 거신후 월간 퍼스널컴퓨터 편집부를 바꿔 아무 기자에게나 문의하시면 됩니다.
그러면 친절하게 당신의 질문에 답해드릴 것입니다.
충분히 이해가 되시면 당신이 알고 계시는 지식이나 알고 싶으신 내용을 원고지나 리포트 용지(타이핑 실력이 뛰어나시면 한글 워드프로세서 등으로 입력하여 디스켓에 수록하셔도 됩니다)에 간략하게 적으신 후
우편번호 140—150
서울시 용산구 갈월동 17
월간 퍼스널컴퓨터 편집부 앞으로 보내시면 됩니다.
게재된 원고는
소정의 고료를 보내드리겠읍니다.

특집

PC통신이란 무엇인가



우리가 흔히 말하는 통신이란 컴퓨터와 주변기기간에 발생하는 데이터통신에서부터 시작하여 컴퓨터와 컴퓨터간의 통신에 이르기까지 그 범위는 매우 넓으며, 종류 또한 다양하다. 이러한 통신은 컴퓨터가 대중화되면서 보급되기 시작하여 현재는 우리들의 생활 깊숙히 파고들 정도로 일반화되었다. 그러나 아직까지도 대부분의 사람들은 이러한 통신에 대해 개념조차 제대로 알지 못하고 있는 실정이다. 따라서 여기서는 데이터통신의 정확한 이해를 위해서 RS-232C란 무엇이며, 컴퓨터간의 접속은 어떻게 하고 어떤 경로를 통하여 통신이 이루어지는지 등 컴퓨터통신의 전반적인 사항에 대해 자세히 설명하고자 한다.

RS-232C

1. RS-232C란 무엇인가?

현재 보급되고 있는 퍼스널컴퓨터에는 대부분 표준장비로서 RS-232C 인터페이스 회로가 마련되어 있다. 또 표준실장이 아닐 경우는 대부분의 컴퓨터 회사에서 옵션으로 공급하고 있다. 이 RS-232C란 도대체 무엇일까?

오늘날에는 대부분의 컴퓨터 소프트웨어들이 패키지화되어 컴퓨터에 대한 기초지식이 그다지 없어도 업무나 연구 또는 오락 등에 컴퓨터를 이용할 수 있게 되었지만 컴퓨터가 많이 보급되지 않은 초기의 경우는 RS-232C란 용어자체가 생소한 것이었다. 따라서 이러한 것은 전문기술자만이 알아야 하는 것으로 인식되기도 하였다.

그러나 요즈음에는 누구나가 흔히 듣는 용어이며 일반적으로 RS-232C란 퍼스컴통신을 위해서 사용하는 기능이라는 것은 대부분의 사람들이 알고 있다. 또 퍼스컴의 뒷면에 코넥터가 나와 있다는 정도는 알고 있지만 어딘지 모르게 가까이 하기 어렵다고 느끼고 있는 사람들이 많으리라 생각한다. 그럼에도 불구하고 퍼스컴통신에 관한 책 등을 펴면 이 RS-232C라는 알 수 없는 용어가 빈번히 나와서 초보자들을 괴롭힌다. 그래서 여기서는 대략적으로 이 RS-232C에 대해서 회로지식이 전혀없는 사람도 이해할 수 있도록 간단히 설명하고자 한다.

왜냐하면 RS-232C 자체가 퍼스컴통신을 아는데에는 아무래도 피해 지나갈 수 없는 관문임과 동시에 가장 중요한 사항이기도 하므로 반드시 이해하고 넘어가야 하기 때문이다.

RS-232C는 1969년 EIA(미국전자공업협회)에서 그 전기신호의 의미나 코넥터의 형상 등 물리적인 규격을 정한 것이다.

여기서 RS는 Recommended Standand 의 약어이고 232는 그 특정규격에 대한 식별번호를 나타낸다. 그리고 C는 RS-232C에 가해진 개정의 상태를 나타내는 말이다.

현재 시리얼전송(1비트씩 보내는 방법)에는 세계적으로도 RS-232C규격이 많이 채용되어 퍼스컴통신의 공통규격으로서 다수의 퍼스컴·유저에게 이용되고 있다는 실정이다.

이 규격은 0~2000bps의 직렬데이터를 전송할 수 있으며 케이블 길이는 50피트로 제한되어 있다.

2. TTY순서

원래 초기의 컴퓨터는 맨머신 인터페이스(인간과 기계와의 커뮤니케이션의 중개를 꾀하는 구조이며, 일반적으로는 하드웨어와 소프트웨어의 양자를 총칭하여 말한다)로서 텔레타이프 등의 입출력장치를 가지며, 인간은 이 입출력장치에 의해 컴퓨터에 지령을 직접 주고 있었다.

이 텔레타이프(일반적으로 TTY라고 표기한다.)도 단순히 컴퓨터에 접속되어 있었던 것은 아니다. 텔레타이프와 컴퓨터간의 케이블에서는 서로의 데이터가 원활하게 주고 받음이 되도록 일정한 약속에 따라서 전기신호가 가기도 하고 오기도 하고 있었다. 이것이 일반으로 TTY순서라는 약속이며 초기의 대표적인 통신순서이기도 하였다. RS-232C인터페이스는 이 연장선상에 있다.

또 RS-232C에 대해서는 커런트 루프라는 규격이 있다. 기계식 텔레타이프에 사용되고 있는 규격이 이 커런트 루프규격이다. 옛날부터 있는 계측기기는 지금에도 커런트 루프밖에 지원하고 있지 않는 것도 있으므로 주의가 필요하다.

그렇다고는 하지만 커런트 루프는 간단한 변환회로로 RS-232C 규격으로 변환할 수 있으므로 어느 정도의 이해만 한다면 두려워할 정도는 아니다.

3. TSS의 구조

RS-232C는 시리얼(직렬)전송의 규격이다. 따라서 송신데이터는 1비트씩 보내진다.

RS-232C가 이렇게 된 것은 퍼스컴을 설계하는 기술자가 TSS(Time Sharing System) 터미널로서의 용도를 퍼스컴에 부가하였기 때문이다.

어떤 기술이나 규격에 그 성장한 내력이라는 것이 있다. 이것을 안다는 것은 그 규격을 이해하는 빠른 길이기도 하므로 여기서 TSS에 대해서 약간의 설명을 하고자 한다.

퍼스컴이전의 초기의 컴퓨터는 고가이고 부피도 커서, 도저히 개인이 소유할 수 있는 것은 아니었다. 큰 기업에서도 풀로 가동시키고 있지 않으면 채산이 맞지 않는다고 하는 시대였다. 이와 같은 시대배경 속에서 태어나서 오늘날에도 가동하고 있는 것이 TSS이다.

TSS란 시분할처리 시스템이며, 간단히 말하면 시간을 잘게 구분하여 많은 단말장치(터미널)의 요구에 조금씩 답하여 가는 방법이다. 조금씩이라고 하여도 인간의 감각에 비하면 무시할 수 있을 정도로 빠르므로 단말측에서 보면 자신이 대형컴퓨터를 점유하고 있는 것 같이 느끼는 것이다.

이와 같은 TSS의 대부분은 통신규격으로서 RS-232C를 채용하고 있다.

4. RS-232의 전기신호와 접속

RS-232C의 전기신호의 주된 것을 표 1에 나타내지만 여기서 모두를 이해할 필요는 없다.

신호에는 각각 이와 같은 의미가 규정되어 있어서 이 규격에 맞게 만들어진 계측기나 기타의 기기와의 접속을 쉽게 할 수 있는 것이다.

그렇다고는 하여도 시판되고 있는 계측기나 모뎀중에는 RS-232C의 규격을 모두 만족하고 있지 않는 것도 있다.

이럴 경우 이와 같은 기기와의 접속을 부득이하게 시키고자 할 경우는 루프 백이라는 수법으로 데이터의 전송

〔표 1〕 RS-232C 각 핀번호의 신호명

핀	신호명		핀	신호명	
①	Protective ground	보안용 접지	14	s. Transmitted data	
②	Transmitted data	송신데이터	15	Transmitter signal element timing	
③	Received data	수신데이터	16	s. Received data	
④	Request to send	송신요구	17	Receiver signal element timing	
⑤	Clear to send	송신가능	18	not defined	
⑥	Data set ready	데이터세트 준비	19	s. Request tosend	
⑦	Signal ground	신호용 접지	⑳	Data terminal ready	데이터 단말준비
⑧	Received line signal detecter	수신 캐리어 검출	21	Signal quality detecter	보수시험
9	not defined		22	Ring indicater	피호표시
10	not defined		23	Data signal rate selecter	데이터 신호속도 선택
11	not defined		24	Transmitter signal element timing(DTE)	
12	s. Received line signal detecter		25	TEST indicater	시험표시
13	s. Clear to send				

S : Secondary—예비나 2대째용　　O : 퍼스컴통신에서 주로 사용하는 것

을 하는 방법이 있다. 루프 백이란 자신이 내고 있는 신호를 상대에게 보내지 않고 자신의 코넥터의 다른 단자에 주고 마치 상대로부터 신호를 받은 것처럼 처리하는 것이다.

극단인 경우에는 그림 1과 같이 그라운드, 송신데이터, 수신데이터의 3개만으로 송신을 하며 다른 단자는 사용하지 않고, 루프 백 시킬 수 있다.

이와 같은 배선에 의해 기기는 데이터통신의 여러 조건을 조사하고 있다고 간주하는 것이다.

접속형태

1. 직접접속과 공중회선접속

퍼스컴과 퍼스컴을 접속하는 형태를 크게 분류하면 다음의 2종류로 나누어진다.

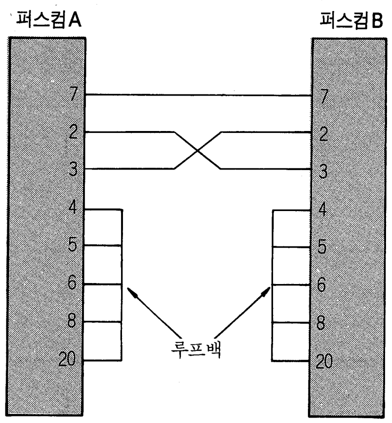
① 퍼스컴과 퍼스컴을 직접 케이블로 접속하는 방법(직접접속)

② 전화회선을 사용하여 퍼스컴간에 데이터를 주고 받는 방법(공중회선접속)

이 2개는 이용되는 형태는 다른 것이지만 원리적으로는 꼭 같다.

요는 2개의 퍼스컴을 연결하는 수단으로서 단지 케이블로 열결하는가, 사이에 전화회선을 경유시켜 원격지까지

〔그림 1〕 루프백



주고받음을 가능하게 하는가 하는 차이 뿐이다.

만일 2개의 컴퓨터를 자유롭게 사용할 수 있다면 우선 이 2대를 짧은 케이블로 연결시키는 것으로 부터 시작하는 것이 통신의 기본원리를 이해하는 가장 좋은 방법이다. 이 방법의 장점은 회선사용료가 필요없다고 하는 것과 발송인과 수신인을 동시에 볼 수 있으므로 어떤 상태일 때 에러가 일어나는가를 곧 피부로 느낄 수 있고 통신의 원리를 단기간에 몸에 익힐 수 있기 때문이다.

짧은 케이블에 의한 직접 연결방법은 통신에 있어서 가장 간단한 방법이므로 이것에 실패한다면 전화회선을 사용한 원격통신을 하기에는 도저히 힘이 든다.

그렇지만 실제로 실험을 하다 보면 직접 연결하는 것도 그렇게 간단하지는 않다는 것을 실감할 수 있을 것이다. 전송을 하다보면 1문자가 빠진다든지 문자의 일부가 예상외로 변하기도 하여 보통수단으로는 안된다.

그러나 이러한 경험은 장래에 대규모의 퍼스컴통신을 할 경우에 중요한 지식으로 될 것이므로 결코 헛된 작업은 아니다.

(1) 직접접속

직접접속이란 한마디로 하면 퍼스컴과 퍼스컴을 직접 케이블로 연결하여 데이터의 송수신을 하는 것이며 송신측과 수신측이 케이블로 연결되는 범위에 있어야 한다. 따라서 직접접속은 신속한 데이터전송을 가능하게 할뿐만 아니라 전화회선을 사용하지 않으므로 회선잡음 및 에러 등이 발생하지 않으며 모뎀에 비해 훨씬 빠른 속도로 전송할 수 있다.

일반적으로 퍼스컴통신이라고 하면 전화회선을 사용한 원격자와의 데이터의 송수신을 연상하지만 전화회선을 사용하지 않고 케이블 직결방식도 훌륭한 통신인 것에는 틀림이 없다. 단지

통신의 수단이 다를 뿐이다.

예를 들면 어떤 정보를 상대에게 전하는데에 큰 소리를 외쳐서 전하여도 되고 봉화를 올려서 전하여도 된다. 또 거울을 사용하여 광의 반사로 전하는 것도 가능하며 전선을 사용하여 전기신호로 전하여도 된다. 이와 같은 전기신호를 사용하는 경우에도 이른바 모르스신호를 사용하는 전신과 직접회화를 할 수 있는 전화가 있다.

어느 것으로 하여도 이들은 모두 수단이 문제이며 정보의 내용과는 무관하다.

직접 접속방식으로 퍼스컴통신을 할 때에 주의해야 할 것은 케이블의 종류이다. 나중에 자세히 설명하지만 크로스타입의 케이블을 사용할 필요가 있다.

이 방식은 서로의 퍼스컴간 거리에 제한이 있지만 그 이외에는 다른 문제점이 없으며 퍼스컴통신을 할 수 있으므로 본격적인 이용을 계획하는 준비단계로서는 큰 의미를 가진다.

(2) 공중회선접속

전화회선을 사용하는 형태에서는 전화를 통하고 있는 곳이면 제아무리 거리가 떨어져 있어도 데이터의 송수신이 가능하다. 이것이 최대의 장점이다. 그러므로 외국과의 데이터송수신도 가능하다. 일반적으로 퍼스컴통신이라는 경우 이 전화회선을 사용하는 케이스를 가리키는 것이 많다.

단, 전화선은 본래 회화를 주체로 한 음성신호를 보내는 것을 목적으로 하고 있으므로 법률적으로도 기술적으로도 전화선에 직접 퍼스컴을 연결하는 것으로는 안된다. 모뎀이라든가 음향커플러라고 하는 것을 거쳐서 전화회선과 퍼스컴은 연결된다.

그 모뎀이나 음향커플러는 퍼스컴에서 나오는 데이터를 전화회선용으로 변환(변조)하여 송출하며 전화회선에서 보내져 온 데이터를 퍼스컴용으로 변환(복조)하여 퍼스컴에 주는 역할을 한다.

특히 모뎀은 종래의 가정 전화제품과는 대단히 성격이 다른 기기이므로 구입할 때에는 주의가 필요하다.

또 음향커플러를 사용하려면 상당한 제한이 따른다.

이것은 음향커플러에 수신기를 세트하기 위한 형상에서 오는 제약이다.

또 음향커플러는 외부의 진동에 의한 잡음 등에 영향을 잘 받으므로 놓는 장소에도 주의하기 바란다. 키보드를 강하게 두드리는 버릇이 있는 사람은 음향커플러는 될수록 멀리두든가 튼튼한 책상을 사용해야 하는 것이다.

2. 모뎀의 규격

여기서는 모뎀의 규격에 대해서 조금 언급해보자.

데이터통신시에는 「CCITT규격」이라는 말에 주의할 필요가 있다.

예를 들면 CCITT(국제전신전화 자문의원회)의 추장규격에도 V.21. V.22, V.23, V.22bis 등이 있다. 이들은 각각 의미하고 있는 내용이 다르다.



이들의 CCITT권고에 대해서는 자세한 내용을 알아 둘 필요는 거의 없지만 모뎀을 선택할 때에는 어느 정도 참고로 해야 한다(표 2).

예를 들어 300bps의 모뎀에서 전2중통신이면 CCITT의 V.21의 규격이면 된다. 그리고 통신하는 상대도 마찬가지의 장치를 가지고 있으면 된다.

또 1200bps의 전2중통신을 하는 것이면 CCITT의 V.22규격이면 된다. 이 경우도 상대가 마찬가지의 장치를 가지고 있으면 통신가능하다.

이와 같은 규격에 대해서는 각 모뎀에 기재되어 있으므로 조사하면 곧 알 수 있다. 따라서 실제로 기기의 선정에 있어서는 주의하기 바란다.

모처럼 고가인 모뎀을 손에 넣어도 상대가 전혀 다른 규격의 모뎀이면 통신은 할 수 없다. 따라서 모뎀을 구입할 때는 호환성문제와 기능, 속도에 따른 제한을 신중히 검토하여 선택하여야 한다.

케이블

1. 코넥터의 형상

RS-232C 코넥터는 2진 직렬데이터, 제어신호 및 타이밍 신호의 전송을 위한 인터페이스 규격에 맞도록 되어 있다.

이러한 접속규격은 DTE(Data Terminal Equipment)와 DCE(Data Circuit Termination Equipment)사이의 물리적인 접속규격에 맞도록 되어 있으며 CCITT, EIA, ISO 등을 비롯한 표준기구들이 표준화를 제정, 권고하고 있다. 왜냐하면 DTE/DCE사이에 어떤 표준규격이 없을 경우, 각자 독자적인 방법으로 DTE/DCE를 생산한다면 커다란 혼란이 발생하기 때문이다.

이와같이 DTE와 DCE사이의 기계적인 면과 전기적인 면, 기능적인 면을 상호접속 처리해주는 장치를 DTE/DCE 인터페이스라고 한다 여기에는

아날로그전송시 이용되는 25핀, 37/9핀과 디지탈 공중데이터 네트워크에 이용되는 15핀 코넥터 등이 있다. 그림 2에 이러한 25핀 코넥터의 핀 배열을 나타낸다. 실제로 DTE와 DCE의 연결은 25핀을 통하여만 되는데 각 상호 접속신호에 대한 핀배열은 사용하는 DCE의 형태에 따른다.

2. 케이블의 종류

퍼스컴과 퍼스컴 혹은 모뎀을 연결하려면 케이블이 필요하다. 일반적으로 케이블에는 스트레이트타입과 크로스타입의 2종류가 있다(그림 3, 4 참조). RS-232C의 케이블을 구입할 때에는 용도에 따라 사용하는 케이블이 다르므로 주의하지 않으면 안된다.

인간에는 음성정보의 전달을 위해서 입(발신기)과 귀(수신기)가 있다. 자신의 입 에서 말한 소리는 상대의 귀에 이르러 상대의 소리는 자신의 귀에 들린다.

퍼스컴도 마찬가지이고 퍼스컴과 케이블을 직접 연결한 경우에는 송신데이터와 수신데이터가 충돌하지 않도록 케이블의 신호선의 일부를 교차시킬 필요가 있다. 이것이 크로스타입의 케이블이다.

한편 인간이 전화로 음성정보를 전달하는 경우는 어떨까? 송신측과 수신측에서 수신기를 꾸미는 방법이 반대로 되는 것은 아니다. 인간과 전화기의 관계를 보면 입은 입, 귀는 귀라는 것 같이 스트레이트에 대응하고 있다. 입에서 나온 음성을 전화기가 자동적으로 상대의 귀에 전하여 준다. 퍼스컴과 모뎀의 관계도 마찬가지이다. 따라서 퍼스컴과 모뎀이나 음향커플러를 연결하는 경우는 스트레이트타입의 케이블을 사용할 필요가 있다.

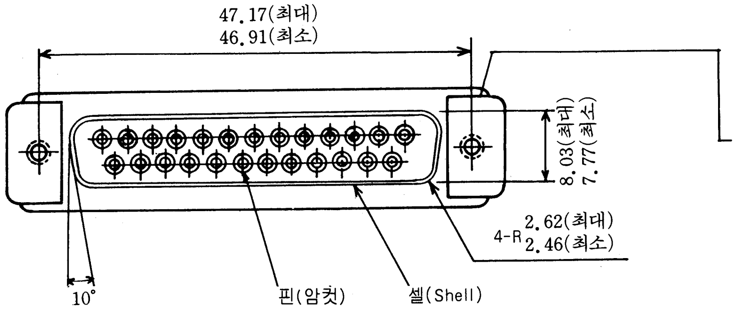
일반적으로 RS-232C통신용 케이블을 전화회선을 사용하는 것이라 생각하여 스트레이트타입으로 되어 있지만 케이블로 퍼스컴끼리를 직접 연결할 경우에는 크로스타입의 케이블을 사용

〔표 2〕 CCITT 규격

V. 21*	일반교환전화망용으로 표준화된 300보오 전2중모뎀, Bell 103으로 호환안됨.
V. 22	일반교환전화망용으로 표준화된 600/1200보오 전2중모뎀, Bell 212로 호환됨
V. 22bis	일반교환전화망용으로 표준화된 2400보오 전2중모뎀, 미국에서도 V. 23bis를 채용
V. 23*	일반교환전화망용으로 표준화된 600/1200보오 반2중모뎀, Bell 2020으로 호환 안됨.
V. 24	데이터 단말장치로 데이터 회선종단 장치간의 상호접촉회로의 정의
V. 25**	수동접속에 반향 서프레서 기능정지를 포함한 일반교환전화망에만 자동기호급을 갖춘 응답장치
V. 26*	4선식 전용회선용으로 표준화된 2400bit/s 모뎀
V. 26bis*	일반교환전화망용으로 표준화된 2400/1200bit/s 모뎀
V. 27	전용회선용으로 표준화된 4800bit/s 모뎀
V. 27bis*	전용회선용으로 표준화된 4800/2400bit/s 모뎀
V. 27ter*	일반교환전화망용으로 표준화된 4800/2400bit/s 모뎀
V. 28	불평형 복류 상호접속회로의 전기적 특제
V. 29*	전용회선용으로 표준화된 9600bit/s 모뎀
V. 36	60~108kHz군 대역회선을 가지고 있는 동기식데이터 전송용 모뎀식

*는 음성대역회선용 DCE(Data Circuit Terminating Equipment : 모뎀등을 말한다)의 상호접속회로번호를 규정한다.
**는 공중전화망용 및 텔렉스용 자동호출장치의 상호접속회로번호를 규정한다.

〔그림 2〕 DEC측 코넥터



하는 것에 주의하지 않으면 안된다.

그렇다고는 하지만 스트레이트타입과 크로스타입의 케이블은 다른 차이가 없고 단지 서너개의 배선을 바꾸는 것만으로 간단히 바꿀 수 있다.

통신속도

퍼스컴통신에서는 전기적인 약속에 따라서 데이터의 송수신을 하고 있다. 따라서 데이터를 보내는 측과 받는 측에서 통신속도가 달라져 있으면 정상적인 송수신을 할 수 없다.

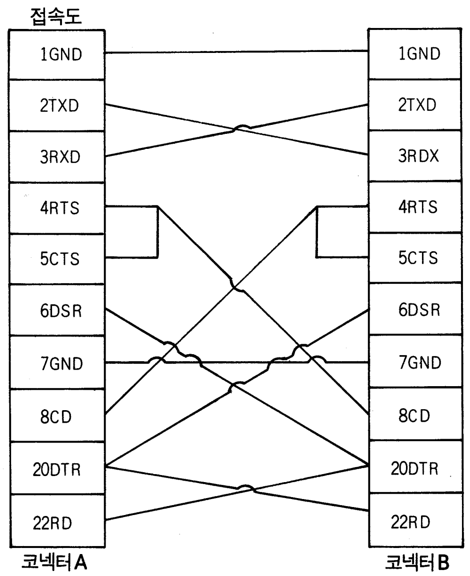
또 최근들어 고속의 데이터통신을 하는 경향이 강해져 오고 있다. 데이터의 고속화는 앞으로 가면갈수록 가속화될 것이다.

1. 통신속도의 결정

그러면 통신속도는 어떻게 하여 결정하는 것인가?

직접접속방식의 경우는 서로간의 퍼스컴의 통신속도가 일치하여 있으면 원칙적으로서 문제는 없다. 전화회선을 사용하여 데이터통신을 할 경우는 서로의 퍼스컴의 통신속도가 일치해야

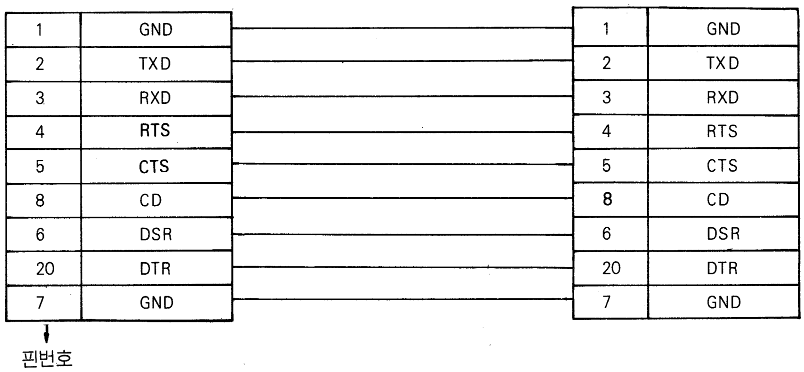
〔그림 3〕 크로스타입 케이블



핀의 접속표

코넥터A	코넥터B
1	1
2	3
3	2
4	A—5 8
5	A—4 8
6	20
7	7
8	4 5
20	6 22
22	20

〔그림 4〕 스트레이트타입 케이블



되고 더구나 모뎀이나 음향커플러의 통신속도도 일치시키지 않으면 안된다. 특히 상대가 여럿인 경우는 가지각색인 조건을 상대마다에 일치시키지 않으면 안된다.

통신속도를 나타내는 단위는 BPS (Bit Per Second)라고 하며 1초간에 보내는 비트수의 의미이다.

일반적으로는 300bps, 1200bps, 2400bps, 4800bps, 9600bps 등으로 하지만 고속인 모뎀은 고가인 것도 있으며, 1200이나 2400bps가 현재 통상 행해지고 있는 퍼스컴통신의 데이터전송속도이다.

그러나 모뎀도 최근은 값이 싼 제품이 시장에 나오기 시작하였으므로 1200bps의 시대에서 2400bps의 시대로 이동하리라 예상된다.

통신방식

1. 전2중과 반2중

통신방식에는 전2중과 반2중의 2가지가 있다. 통신은 본질적으로 보내는 쪽과 받는 쪽이 있다.

더구나 이 2가지의 역할은 고정되지 않은 경우가 많다.

사람들이 서로 이야기할때는 랜덤하

□ 컬럼

BPS와 보오

데이터의 신호속도의 단위에 BPS라든가 비트/초라고 하는 표현을 사용한다. 그런데 보오라고 하는 단위도 있지만 이것은 데이터의 변조속도이며 1엘레먼트를 1초간에 보내는 속도를 보오라고 한다. 그러므로 주로 모뎀의 성능을 말할 때에 2400보오 모뎀 등이라 표현한다.

변조속도를 알기 쉽게 말하면 컴퓨터에서 출력되는 디지탈신호를 아날로그신호로 변화시키는 속도이므로 일반적으로는 보오와 BPS는 일치한 값이 된다.

그러면 데이터신호속도에는 어느 쪽의 단위를 사용하여도 되는가 하면 그렇지는 않다.

BPS와 보오의 값이 다른 예를 소개하겠다.

모뎀의 변조방식에도 몇가지나 있으며 진폭변조방식(AM)이나 주파수변조방식(FM)은 변조속도와 데이터신호속도가 같아지지만 위상변조방식에서는 변조의 1엘레멘트가 2비트(4상식)나 3비트(8상식)가 일반적이므로 BPS와 보오의 단위에 의해서 다른 값으로 된다.

예를 들면 4상식 위상변조에서는 1변조에서도 2비트전송하므로 변조속도를 1200보오로 하면 데이터신호속도는 2400비트/초로 된다.

따라서 변조속도는 보오, 데이터신호속도는 비트/초를 사용하는 것이 올바르다.

게 이야기를 하는 사람과 듣는 사람이 바뀐다. 역할이 고정되어 있는 예에서는 연설을 하는 사람과 청중의 경우를 들 수 있다. 통신방식이란 이 보내는 쪽과 받는 쪽의 전환을 어떻게 하는 것인가 하는 문제이다. 요컨데 전화와 같이 이야기하면서 듣는 방식인가 무선기와 같이 이야기할 때와 들을 때를 명확하게 구별하는 방식인가의 차이이다.

무전기와 같은 일방통행의 주고받음을 반2중이라하고 양쪽방향으로 동시에 전송이 가능한 경우를 전2중이라 한다.

2. 단향통신

이상과 같이 전2중통신과 반2중통신 이외에도 단향(Simplex)통신이 있다. 이것은 퍼스컴통신에서는 거의 사용되는 일은 없지만 구체적인예를 들어 보자.

예를 들어 증권회사를 생각해 보자. 증권회사를 가면 플로어의 벽 가득히 회사의 명찰과 숫자가 전광표시되어 있다. 이것은 주가의 데이터를 서비스하는 회사에서 각 증권회사에 데이터를 일방적으로 보내어 현재의 주가추이가 반영되고 있는 것이다. 따라서 증권회사의 표시판은 수신전용이며 데이터서비스회사의 장치는 송신전용이다. 이것이 단향통신이다.

데이터통신은 아니지만 라디오·텔레비전이나 유선방송 등의 형태도 이 부류에 들어 간다.

3. 2선식과 4선식

이들의 통신방식에도 2선식과 4선식이 있다.

일반적으로 전기는 2개의 전선이 없으면 통하지 않는다. 그러므로 2개의 전선이 연결되어 있으면 전기신호의 주고 받음이 가능하게 된다.

냉장고나 청소기의 전원선이나 스테레오 스피커의 뒤 배선을 보면 이해가 될 것이다.

전화기도 마찬가지로 1회선에 2개의 도선이 있다. 따라서 반2중통신을 하려면 전화기1대(2선식)만 있으면 된다.

반2중의 경우는 2선식이므로 어느 쪽인가가 데이터를 전부 수취하는 것에 전념하여 상대로부터「이제 보내도 된다」고 하는 신호를 수신한 후 비로소 데이터의 송신이 가능하게 된다.

한편 전2중통신을 하려면 전화기가 2대(2선식이 2조인 4선식) 있으면 된다.

4선식의 전2중통신은 한쪽의 2선을 송신용, 다른 쪽의 2선을 수신용으로 사용하므로 동시에 데이터의 주고받음이 가능하게 된다.

그러나 4선식의 전2중통신을 하려면 전화회선이 2회선이 필요하다.

그렇지만 전화기는 2선식에도 불구하고 서로 동시에 이야기할 수 있다. 이것은 전2중통신과 같다. 2선식에서도 전2중통신을 할 수는 없을까?

이런 점을 감안하여 착상한것이 2선식 전2중통신 방식이다. 즉 요즈음의 퍼스컴통신에서는 2선식의 전2중통신이 잘 사용되고 있다. 현재 전화회선을 이용하여 이루어지고 있는 퍼스컴통신의 거의 이 방식이다.

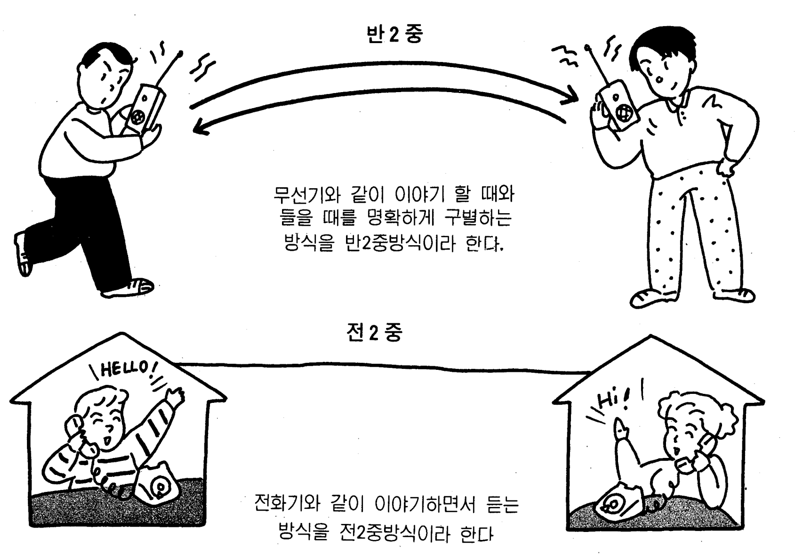
4. 2선식 전2중통신의 원리

간단하게 원리를 소개하면 수신측과 송신측에서 다른 주파수를 사용하여 주파수분할을 하는 것이다. 물론 퍼스컴의 오퍼레이터는 그런 것을 알 필요는 없다.

그러나 이와 같은 2선식 전2중통신이 가능한 모뎀이나 모뎀이 붙은 전화 및 음향커플러에는 송신측으로 되는가 수신측으로 되는가의 구별을 하는 스위치가 있다. 통신은 송신측이 CALL, 수신측이 ANS라고 하는 표시로 되어 있다. 또 자동적으로 전환하는 AUTO 기능을 가지는 것도 있다.

이 스위치는 미리 상대와 약속하여 정하여 두면 어느쪽이 어느쪽이든 좋다. 즉 사용하는 주파수대를 선택하는 것만의 이야기이며 같은 설정으로 되어 있지 않으면 되는 것이다.

퍼스컴 네트워크 등에서는 네트워크가 ANS측이며 퍼스컴측에서는 CALL로 하여 두는 것이 일반적이다.



송신부호

데이터통신에서 전송하는 부호에 대

해서 이전에도 약간 언급하였지만 여기서 종합하여 설명한다.

컴퓨터의 초기시대에는 각 계산기메이커가 컴퓨터설계시에 제멋대로 부호를 만들고 있었지만 점차로 호환성의 문제가 나왔으므로 국제적으로도 부호의 표준화가 진행되어 왔다.

1. ASCII코드

정보교환용의 미국 표준코드로서는 ASCII(America Standard Code for Information Interchange)코드가 있으며, 이것은 0에서 127까지의 번호가 문자, 숫자, 구둣점 및 특수문자에 할당되어 있다(표 3 참조).

이 아스키코드값으로 데이터를 전송할 때는 주의하여야 한다. 왜냐하면 0에서 127까지의 값을 갖는 아스키문자는 대부분의 컴퓨터가 같은 코드를 사용하지만, 128이상의 값을 갖는 아스키코드는 기종에 따라 조금씩 틀리다.

EBCDIC코드

한편 정보교환용 부호로서 대형컴퓨터나 미니컴의 일부에서는 EBCDIC코드가 사용되고 있다.

EBCDIC코드는 8비트코드의 일종이다. 그러나 이 EBCDIC코드가 전혀 퍼스컴과 관계가 없는 것도 아니다. 예를 들면 퍼스컴의 데이터를 대형컴퓨터에서 이용한다든가, 반대로 대형컴퓨터의 데이터의 일부를 퍼스컴에 도입하여 이용할 경우에는 반드시 필요하다.

동기방식

통신을 할때는 서로의 송신 수신속도가 일치하지 않으면 안된다고 하였는데 실은 그것만으로는 아직 충분하지는 않다.

이것을 충분히 하기 위해서는 동기방식이 일치하여야 한다.

1. 전송방식

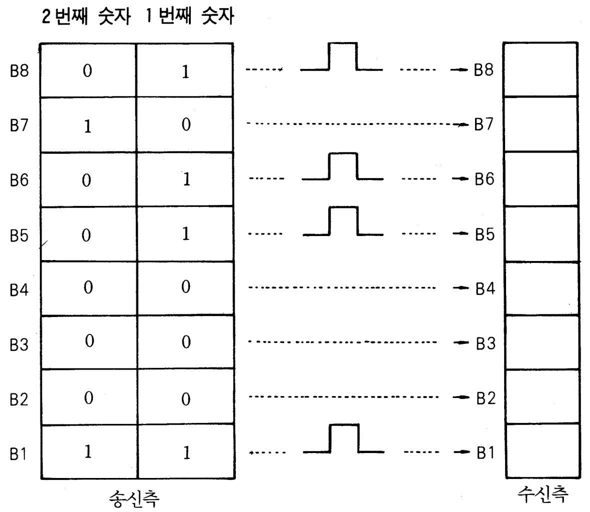
데이터의 전송에는 직렬전송과 병렬

〔표 3〕 ASCII코드표

BIT NUMBERS															
							b7 →	0	0	0	0	1	1	1	1
							b6 →	0	0	1	1	0	0	1	1
							b5 →	0	1	0	1	0	1	0	1
b7 ↓	b6 ↓	b5 ↓	b4 ↓	b3 ↓	b2 ↓	b1 ↓	HEX1 / HEX0	0	1	2	3	4	5	6	7
			0	0	0	0	0	NUL	DLE	SP	0	@	P	·	p
			0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
			0	0	1	0	2	STX	DC2	"	2	B	R	b	r
			0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
			0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
			0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
			0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
			0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
			1	0	0	0	8	BS	CAN	(	8	H	X	h	x
			1	0	0	1	9	HT	EM	)	9	I	Y	i	y
			1	0	1	0	A	LF	SUB	*	:	J	Z	j	z
			1	0	1	1	B	VT	ESC	+	;	K	[	k	{
			1	1	0	0	C	FF	FS	,	<	L	\	l	¦
			1	1	0	1	D	CR	GS	—	=	M	]	m	}
			1	1	1	0	E	SO	RS	.	>	N	^	n	~
			1	1	1	1	F	SI	US	/	?	O	—	o	DEL

NUL : 널
SOH : 헤드작업 개시
STX : 텍스트 개시
ETX : 텍스트 종료
EOT : 송신 종료
ENQ : 조합
ACK : 액크놀리지(정논리)
BEL : 벨
BS : 1문자 후진
HT : 수평탑
LF : 개행
VT : 수직탑
FF : 폼피드
CR : 캐리지 리턴
SO : 시프트 아웃
SI : 시프트 인
DEL : 데이터 링크 중단
DC : 디바이스제어
NAK : 액크놀리지(부논리)
SYN : 동기 아이들링
ETB : 송신 블록 종료
CAN : 캔슬
EM : 중간 종료
SUB : 치환
ESC : 이스케이브
FS : 커서라이트
GS : 커서레프트
RS : 커서업
US : 커서다운
SP : 스페이스(블랭크)
DEL : 소거

〔그림 5〕 병렬전송



송신측이 핀101(B1)이라면 수신측도 (B1), 송신측이 핀101(B2)이라면 수신측도 (B2), 이와 같이 비트마다 그대로 송신하는 방법이다.

전송이 있으며, 병렬전송은 8단위 부호이면 8단위를 동시에 송신하고, 수신하는 쪽도 8단위 동시에 수신한다고 하는 사정으로 데이터를 병렬로 보내는 방법이다(그림 5 참조).

따라서 전송속도는 빠르지만 컴퓨터 내부의 출력용 레지스터를 포함하여 전송로도 각 비트분이 필요하게 되어 전송경비가 드는 결점을 가진다. 퍼스컴용 프린터의 표준규격으로 되고 있는 센트로닉스규격은 병렬전송의 예이다.

한편 직렬전송은 데이터를 1세트씩 순번로 보내는 방법이며, 단순하여 시간은 걸리지만 전송경비가 싸게 들므로 현재의 퍼스컴통신에 한하지 않고 데이터전송은 일반적으로 이 방법을 사용하고 있다. (그림 6 참조).

2. 동기방식의 종류

통신회선에서 보내져 온 데이터는 송출측 어느 타이밍을 잡으면서 데이터를 받아들이지 않으면 안된다.

이 타이밍을 잡는 것을 동기라고 한다.

동기란 송신측에서 송출한 데이터가 에러없이 전화회선(전송로)에 실려서 수신측에서 데이터를 확실하게 수취하기 위한 중요한 약속이다.

동기방식에는 (1) 조보동기 (2) 독립동기 (3) 프레임동기 3가지가 있다.

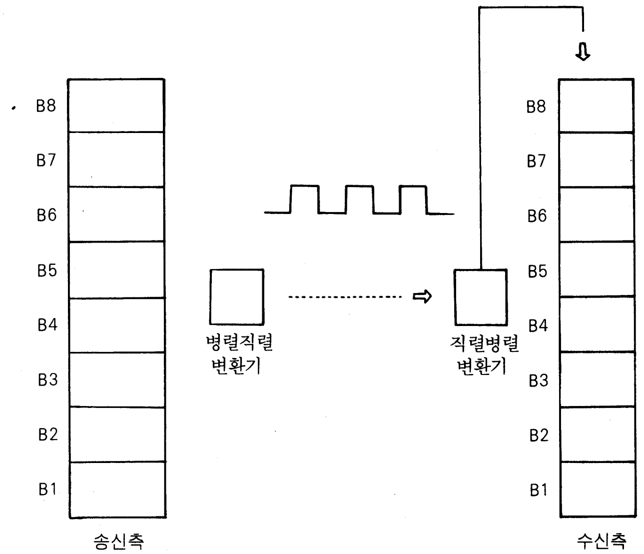
(1) 조보동기

조보동기방식은 그 간편함 때문에 300~1200bps의 퍼스컴통신에서 가장 널리 행해지고 있는 동기방식이다.

조보동기의 특징은 송출하는 데이터(1문자)의 선두에 스타트비트(ST)를 붙이고 그 데이터의 말미에 스톱 비트(SP)를 붙인다.

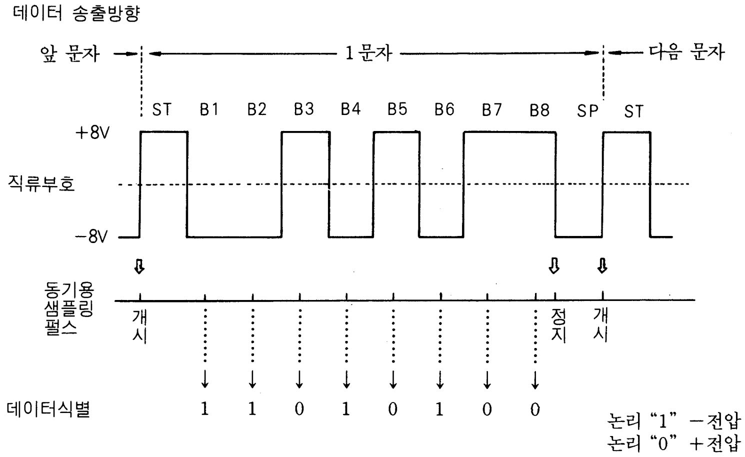
각 문자마다에 적어도 2비트 씩을 부가하여 데이터를 보내므로 전송효율은 나빠지지만 원리가 간단하므로 퍼스컴통신에서는 널리 행해지고 있다. 이른바 무순서통신이란 이 방식이다(그림 7 참조).

〔그림 6〕 직렬전송



1번째 숫자의 핀1이 (B1)에서 (B2)……→(B8), 이와 같이 8단위의 병렬데이터를 비트마다 연속적으로 직렬데이터로 변환하여 송출한다. 수신측에서는 반대로 수신된 직렬데이터를 (B1)에서 차례로 병행하여 교체하여 8단위의 병렬데이터로 구성한다.

〔그림 7〕 조보동기



(2) 독립동기

독립동기는 송신데이터의 문자열 앞에 동기부호(SYN : 싱크로의 의미)를 2개 이상 연속하여 부가해서 전송하는 방법이다.

수신측에서는 동기부호를 수취하면 동기부호를 근본으로 동기를 잡고 이후 일반으로 8비트씩 구분하여 데이터를 조립한다. 따라서 대량의 데이터를 전송하는 데에 적합한 방법이다.

(3) 프레임동기

프레밍동기는 HDLC(High Level Data Link Control)전송제어 순서로 사용된다.

그 원리는 프레임의 선두와 말미에 FLAG라고 불리우는 특정의 비트 패턴을 부가하여 동기를 잡는 방법이다.

새로 선보인 한글 워드프로세서 '혼글'

새로운 워드프로세서가 3명의 학생(이찬진 : 서울대 기계과졸, 김형집 : 서울대 전자과 3, 우원식 : 서울대 제어계측과 3)들에 의해 개발되어 관심을 끌고 있다.

이번에 개발된 워드프로세서 '혼글'은 그 명칭에서도 볼 수 있는 바와 같이 고어의 입력도 가능한 새로운 스타일의 워드프로세서로서 다양한 기능들을 가지고 있어 지금까지 나온 워드프로세서들의 장점만을 집약시켜 놓은 느낌을 가지게 한다. 앞으로 많이 사용되면 그 장단점을 알 수 있겠지만 여기서는 기존의 워드프로세서와 다른 특징만을 골라 간략하게 소개한다.

한글과 영문자, 일본어 뿐만이 아니라 고어의 입력까지 가능하고 4888자의 한자도 지원되며 문자체의 종류도 명조체와 고딕체 및 이탤릭체와 필기체까지 표기가 가능한 한글 워드프로세서가 나와 장안의 화제가 되고 있다. 기존 워드프로세서의 장점만을 따와 만들어진 '혼글'은 그만큼 다양한 기능을 가지고 있으며 무엇보다도 유저의 입장을 고려하여 사용하기에 편리하게 만들어졌다는 것이 특징이다.

입력가능한 문자

(1) 한글

'혼글'은 현대어는 물론 화면 1에 나타난 바와같이 국문학적으로 가치있는 고어까지 입력할 수 있는 특징을 가지고 있다. 또한 타 워드프로세서와는 달리 한글입력시 Capslock키의 상태를 고려할 필요가 없으며, 한글입력과 상관없이 콘트롤키가 작동하므로 에디트하기 위한 키보드의 영문과 한글전환이 필요없다. 그리고 2벌식, 3벌식 공병우 자판을 사용할 수 있어 어떤 타자기를 사용하던 사람이라도 이 워드프로세서를 비교적 쉽게 사용할 수 있다.

(2) 한자

정부에서 표준으로 정한 KSC-5601-1987에 따른 4888자의 한자를 제공한다. 한글자씩 입력할 때에는 빈도순에 의해 배열된 한자중에서 원하는 글자를 쉽게 선택할 수 있으며, 이와 함께 20,000단어의 한자사전을 통해 자동적으로 단어단위의 한자변환을 할 수 있다. 만약 한자사전에 등록되지 않은 글자는 자동으로 등록된다.

(3) 영문자

영문자는 일반적으로 26자의 알파벳을 기준으로 나라에 따라 코드의 배열이 다른데 이 외에도 외국어를 사용할 수 있도록 미국의 영문자는 물론 불어, 독어, 영국어, 이태리어, 덴마크/노르웨이어, 스웨덴어 등의 입력이 가능하다.

그리고 모든 알파벳의 자판배열은 IBM-PC의 International Character Set규칙에 맞추어져 있어 편리하게 입력할 수 있으며, 사용자가 필요한 다른 국가의 알파벳도 추가할 수 있다(화면 2 참조).

(4) 일본어

히라가나와 가타카나를 표준 일본어 자판에 의해서 입력할 수 있고 일어에 익숙하지 못한 사람을 위해 일어의 영어발음에 의해 입력하는 것도 가능하다.

(5) 그래픽문자

① **전각**

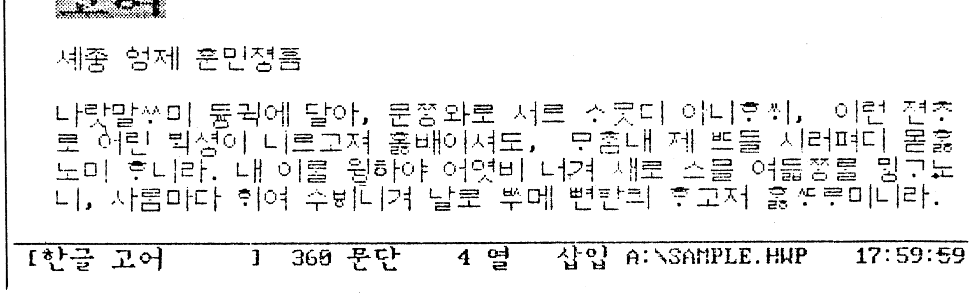
총 31벌 ✕ 94자=2914자 까지의 그래픽 문자가 제공되어 많은 수의 그래픽문자를 사용할 수 있으며, 그중의 1벌은 사용자가 직접 정의할 수 있다. 다음 화면 3은 그래픽문자(전각)의 예를 나타낸다.

② **반각**

한글과 같은 크기의 전각특수문자의 제공과 함께 반각의 특수 문자가 제공되어 여러 곳에 이용될 수 있다.

도형문자로 글을 장식할 수 있고, 그리스문자, 수학문자Ⅰ, 수학문자Ⅱ가 제공되어 수학이나 과학식에 이용할 수 있으며, 화학문자를 이용하여 화학식이나 분자식에도 이용할 수 있다. 또한 러시아 문자인 Cyrillic 문

〔화면 1〕



자도 제공된다.

(6) 선문자 그리기

문서의 양식을 작성할 때 그릴 수 있는 선의 종류로 7가지가 제공되어 다양한 문서를 작성할 수 있다.

[화면 2]

```
USA                 # $ @ [ \ ] ^ ` { | } ~
France              # $ à ° ç § ^ ` é ù è ¨
Germany             # $ § Ä Ö Ü ^ ` ä ö ü β
United Kingdom      £ $ @ [ \ ] ^ ` { | } ~
Denmark/Norway      # $ @ Æ Ø Å ^ ` æ ø Å ~
Sweden              # ¤ É Ä Ö Å Ü é ä ö å ü
Italy               # $ @ ° \ é ^ ù à ò è ì
```

문자꾸미기

WYSIWYG적인 특징을 가지고 있어 옆으로 두배확대, 위아래 두배확대, 진하게, 밑줄, 역상, 음영, 그림자, 외각선 등 여러속성을 글자마다 조합하여 사용할 수 있다. 전체적으로 256(2^8)가지를 사용할 수 있다.

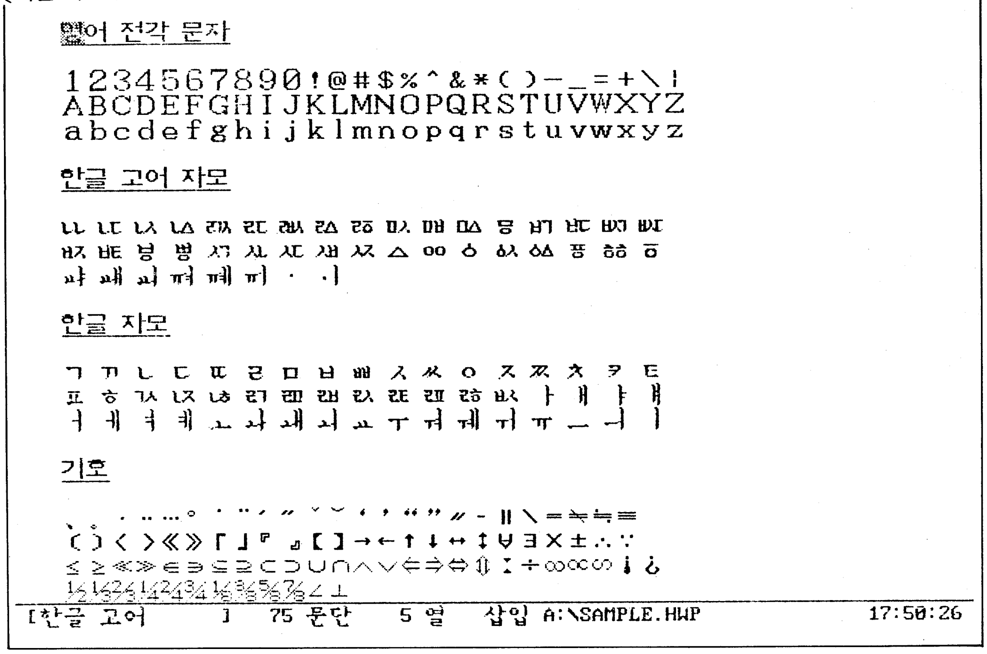
문자체 종류

한글과 일본어는 각각 명조체와 고딕체가 제공되며 영문자에서는 윗첨자와 아래첨자, 이탤릭체와 필기체 등의 다양한 글자체가 제공된다.

자동 문장정렬기능

각각의 문단(Paragraph)마다 좌측마진, 문단 폭 설정기능과 정렬방식이 제공되어 한번 주어지면 특성들이 계속 유지되어 사용자가 직접 문장정렬할 필요가 없다.

[화면 3]



문단을 정리할 때도 문단속의 단어들이 행의 끝에서 잘리지 않으면서 좌, 우, 중앙에 정렬하는 방법이 있고 문단의 양쪽에 정렬하는 방법도 있다.

또한 새로운 문단이 시작할 때 그 첫번째 줄만 들어가거나 나올 수 있도록 들여쓰기와 내어쓰기가 자동으로 처리된다.

워드프로세서 '혼글'에서는 각 줄의 간격을 점단위로 조절하여 가장 알맞는 간격을 유지할 수 있으며, 이러한 기능과 특수기호를 이용하여 화학식이나 수학식을 자유롭게 입력할 수 있다. 보통 이 줄간격 기능은 수학식외에도 문서의 모양을 보기좋게 할때도 쓰인다(화면 4 참조).

이 정렬기능은 정지기능이 있어 입력을 모두 끝낸 후 일괄적으로 정렬할 수 있고 이 기능을 사용하게 되면 입력하는 대로 정렬되어 화면에 표시된다.

그리고 타자하는 과정에서 문단 사이를 쉽게 구분할 수 있도록 컴퓨터가 재정렬하는 과정에서 생긴 공백을 구분하여 표시하도록 할 수 있고, 이 기능을 원하지 않는다면 정지시킬 수도 있다.

기타 특징

① 타자수에 따른 자동저장기능을 두어 0~5000타마다 입력한 내용이 자동적으로 디스크에 저장되어 데이터를 잃어버리는 사고를 방지할 수 있다. 미리 지정된 타수는 500타이며, 사용자가 적절하게 지정할 수 있다.

② 윈도우를 나눌 수 있어 동시에 2개의 서로 다른 화일을 비교하면서 편집할 수 있다. 윈도우는 수직과 수평 두가지 방법으로 나눌 수 있다.

③ 키보드 매크로기능과 상용구 저장기능으로 키입력의 효율을 높일 수 있다.

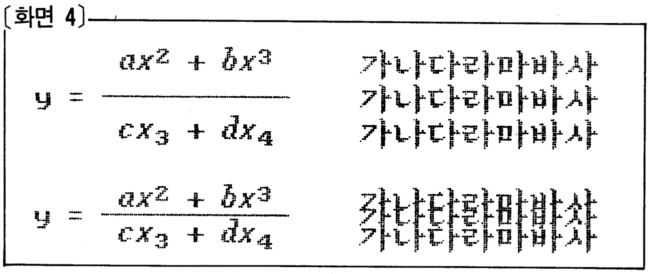
④ 각각의 입력모드에서 현재 자판배치를 보여주는 '자판배열'기능으로 일어, 특수문자의 배치를 알 수 있다.

⑤ 단축키(Short Cut)가 있어 어느 상태에서도 필요한 기능을 선택할 수 있어 빠르게 편집할 수 있고 사용자가 단축키를 정의할 수 있다.

⑥ 사용자가 누르는 키를 최우선으로 처리하기 때문에 사용자는 기다리는 일이 없다.

⑦ 대부분의 명령어가 워드스타, 터보에디터와 호환성이 있다.

〔화면 4〕



⑧ 최근에 편집한 화일의 이름을 기억하고 있어 편집하는 화일을 손쉽게 찾아 편집할 수 있고, 각각의 화일에 최후의 커서위치를 기억하므로 편리하다.

⑨ 윈도우와 팝업 메뉴를 통한 편리한 사용자 인터페이스를 이루고 있어 초보자도 쉽게 이용할 수 있다. 또한 메뉴키를 누르고 잠시 기다리면 그 메뉴에서 사용할 수 있는 기능들이 다시 윈도우로 나타나 짧은 시간안에 익숙해 질 수 있다.

⑩ 보석글의 자료와 팔란티어의 화일, 일반 텍스트화일을 읽어들여 데이터를 교환하여 사용할 수 있다.

⑪ Dos Shell기능이 있어 문서를 편집하는 중간과정에서 도스로 빠져나와 다른 작업을 할 수 있다.

⑫ 시계, 계산기 등의 액세서리가 있다.

이 외에도 다양한 기능들을 가지고 있어 현재 나와있는 워드프로세서인 보석글, 팔란티어 워드프로세서, 한글2000 등과 함께 좋은 경쟁이 되리라 생각하며 판매예정가격도 4만원으로 책정되어 있어 기존의 워드프로세서에 비해 저렴하여 사용자의 관심을 끌 것으로 기대된다.

여러가지 워드프로세서가 이용되고 있는 현시점에서 새로운 워드프로세서인 '훈글'이 개발되어 더욱 편리한 워드프로세싱 작업을 제공하는 것은 반가운 일이다. 그러나 이러한 작업이 컴퓨터 전문 생산업체에 의해 이루어지지 않고 사용자입장인 학생들의 손에 만들어졌다는 사실은 웬지 허전함을 느끼게 한다. 4월초 시판예정인 한글 워드프로세서인 '훈글'에 대한 문의는 러브리소프트웨어 (02)272-1337로 하면 된다.

레지스터와 메모리

김정선 /한국항공대학 전자과 교수

순서논리회로의 구성은 1개 이상의 플립플롭과 0개 이상의 조합논리회로(게이트 회로)로 구성된다. 즉 순서논리회로는 게이트회로가 없더라도 반드시 플립플롭회로가 1개 이상 있어야 한다. 순서논리회로의 일종인 레지스터 역시 최소한 1개 이상의 플립플롭회로가 반드시 존재하여야 한다.

이와 같이 플립플롭이 기본이 되는 레지스터는 그 명칭이 제시하는 바와 같이 플립플롭의 기억 특성을 이용한 정보 기억장치의 일종이다. 따라서 하나의 플립플롭회로는 1이나 0과 같은 2진수의 1비트 정보를 기억할 수 있는 1비트 레지스터라 할 수 있으므로 N개의 플립플롭으로 구성된 레지스터는 N비트의 정보를 저장시킬 수 있는 기억회로가 되는 것이다.

N개의 플립플롭으로 구성되어 있는 N비트의 레지스터 중에 MSI형태가 시중에서 가장 쉽게 구입할 수 있으며, 그 구성형태도 다양하여 각종 디지탈 시스템을 설계 제작하는데 편리하다.

레지스터는 각 비트(즉, 각 플립플롭)간에 연관성을 갖지 않고 독립적으로 동작하며, 단순히 정보의 기억만을 목적으로 하는 레지스터와 각 비트간에 연결성을 가지고 있어 기억된 정보형태를 클럭펄스에 의해 순차적으로 변형(혹은 이동)시킬 수 있는 시프트레지스터로 크게 구분된다. 그러므로 레지스터는 저장된 정보의 비트가 이동하는 형태에 따라 왼쪽 시프트레지스터와 오른쪽 시프트레지스터로 구분한다.

또한 정보의 입력 및 출력방법에 따라 병렬입력 / 병렬출력(Parallel Input / Parallel Outpur : PIPO) 레지스터, 병렬입력 / 직렬출력(Parallel Input / Serial Output : PISO) 래지스터, 직렬입력 / 병렬출력(Serial Input / Parallel Output : SIPO)레지스터, 직렬입력 / 직렬출력(Serial Input / Serial Output : SISO) 레지스터로 구분한다.

레지스터

레지스터와 매우 유사한 것으로 래치(Latch)회로가 있다. 래치와 레지스터는 입력정보의 기억동작이 클럭펄스의 지속(1 또는 0이 유지되는 기간) 기간에 이루어지는 경우를 래치회로라 하며 이 펄스의 지속기간(Pulse Duration) 동안에 동작하는 회로를 게이트화된 래치(Gated Latch)라고 한다. 이러한 래치회로를 피드백 경로를 갖는 순서논리회로에 사용하는 것은 적합치 않다.

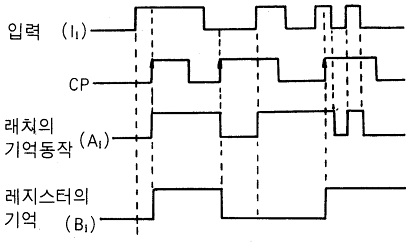
그러나 래치회로 역시 정보를 일시 저장하는 기능을 갖고 있으므로 일종의 레지스터라고 할 수 있다. 래치회로는 그림 1과 같이 D형 플립플롭으로 구성할 수 있다.

레지스터는 클럭펄스의 전이(Transition)에서 정보의 기억동작이 이루어진다. 따라서 레지스터회로는 에지트라거형의 플립플롭이나 주종 플립플롭(Master-Slave Flip Flop)형으로 구성되며, 궤환경로를 갖는 순서논리회로 설계에 적합하다.

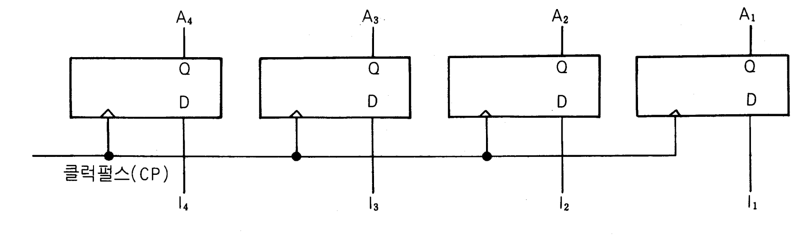
이들 두 회로의 동작상 차이점을 그림 2에 나타내었다. 그림 2에서 래치의 동작은 CP가 1인 상태가 끝나는 시점에서 입력상태가 기억되며, 레지스터는 CP가 0에서 1(혹은 1에서 0)로 변화하는 시점에서 입력상태가 기억된다.

레지스터의 기억내용은 플립플롭의 출력 즉 Q단자의 상태로서 알 수 있다.

〔그림 2〕 래치와 레지스터 동작의 차이



〔그림 1〕 래치회로



말하자면 레지스터의 모든 비트 출력이 0(혹은 1로 하는 경우도 있다)일 경우는 기억내용을 지운다(Register Clear)는 내용이다.

이와 같이 레지스터의 모든 내용을 동시에 0으로 하기 위해서는 플립플롭의 직접 리세트단자(RD)를 사용하면 편리하다. 직접 리세트단자의 입력은 클럭펄스에 무관하게 플립플롭의 출력을 0으로 할 수 있기 때문이다.

그림 3에 보인 바와 같이 모든 플립플롭의 리세트단자를 병렬로 연결하여 클리어를 입력하면 모든 출력은 동시에 0이 된다. 그리고 연속적으로 1과 0을 반복하는 클럭펄스의 입력에 대하여 임의의 시간에 입력된 입력신호(I_1) 상태만을 기억시키고 이 상태를 임의의 시간동안 또 다른 입력신호나 클럭펄스에 무관하게 유지시키기 위하여 그림 3에서와 같이 로드신호를 입력하는 회로를 구성할 수가 있다.

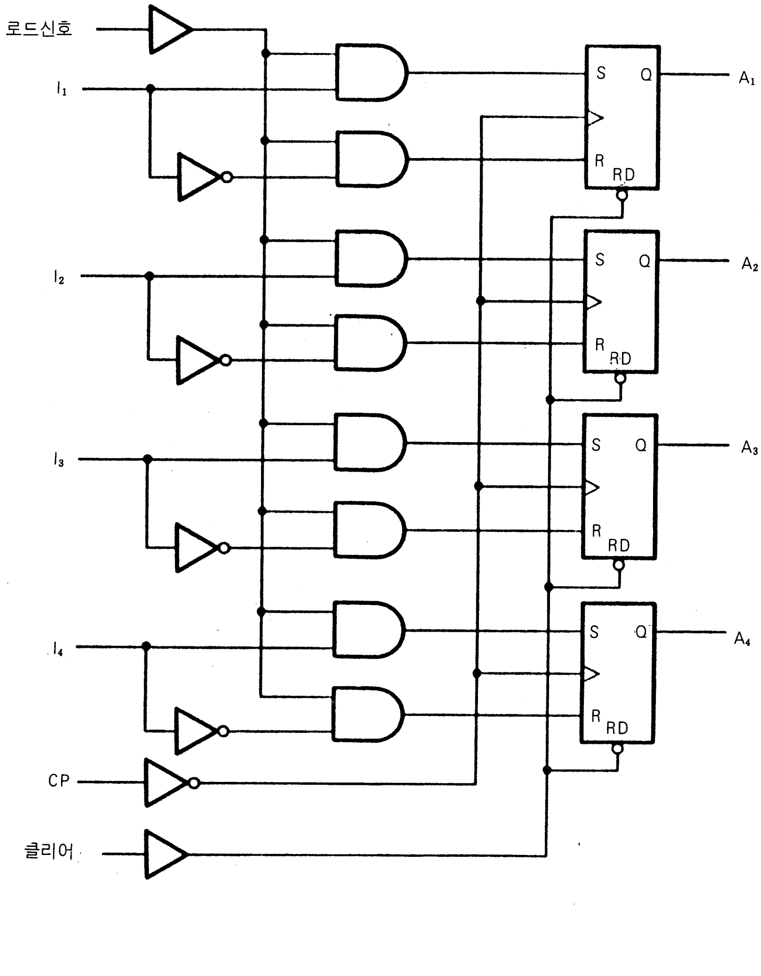
그림 3에서 로드신호 입력, CP입력, 클리어 입력측에 있는 버퍼나 인버터는 각 입력신호에 대한 부하를 줄이기 위하여 필요한 것이다. 그림 3의 회로에서 로드신호는 1일 때 I_1, I_2, I_3, I_4의 입력상태가 각 플립플롭의 S, R에 AND게이트를 통하여 전달되며, 이때 클럭펄스의 1에서 0으로 변화하는 부에지(Negative Edge)에서 기억동작이 이루어진다. 로드신호가 0인 동안은 또 다른 I_1~I_4의 입력이나 클럭펄스의 변화에도 기억상태는 변하지 않고 그대로 유지된다. 그리고 클리어 단자가 0으로 되면 모든 플립플롭은 리세트되며, 레지스터는 클리어된다. 이 회로의 동작상태를 그림 4에 나타내었다.

이와 같이 레지스터는 임의의 시간에 정보를 취하여 원하는 시간만큼 기억상태를 유지시킬 수 있는 회로이며 이러한 동작과 특성은 디지탈 시스템에서는 대단히 중요한 역할을 하는 부분이다. 컴퓨터나 디지탈 시스템에서 이러한 레지스터는 4비트, 8비트, 16비트, 32비트 등으로 구성하여 이용된다.

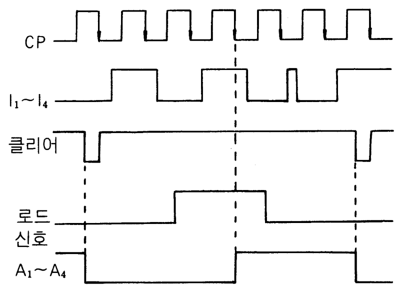
●시프트 레지스터

내부에 저장된 정보를 왼쪽이나 오른쪽으로 자리이동시킬 수 있는 레지스터를 시프트 레지스터라고 하며 그림 5에 일예를 보였다. 그림 5에서 입력 SI는 J_1, K_1에 서로 역으로 입력되며, 클럭펄스가 가해질 때 Q_1, $\overline{Q_1}$가 출력되고, 동시에 이 출력은 J_2, K_2에 가해진다. 마찬가지로 다음 주기의 클럭펄스가 가해질 때 J_2, K_2상태는 Q_2, $\overline{Q_2}$에 전달되며 동시에 J_3, K_3에 가해진다.

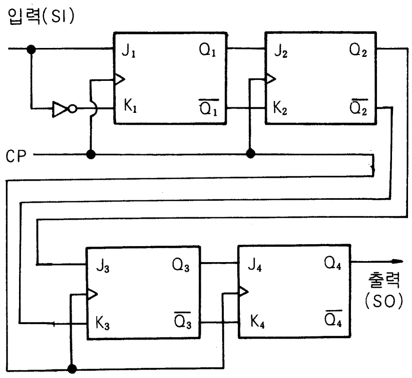
〔그림 3〕 병렬입력이 가능한 4비트 레지스터



〔그림 4〕 레지스터의 동작 파형



〔그림 5〕 4비트 오른쪽 시프트 레지스터



같은 방법으로 SI에 가해진 입력은 클럭펄스 수만큼 SO쪽으로 이동하여 나타나게 된다. 그림 5에서는 정보의 이동방향이 왼쪽에서 오른쪽으로 나타나므로 오른쪽 시프트 레지스터라고 한다. 만약 그림을 반대로 하여 오른쪽에 입력단자, 왼쪽에 출력단자가 배치되도록 하였다면 왼쪽으로 정보의 자리이동이 이루어지므로 왼쪽 시프트 회로가 될 것이다.

클럭신호 입력단자에 AND게이트를 사용하여 그림 6과 같이 구성하였을 경우, 시프트 제어신호가 1일 경우만 클럭펄스가 가해지므로 이 때만 시프트가 이루어지게 된다.

이와 같은 시프트 레지스터는 원하는 시간에 원하는 비트수만큼 자리이동을 시켜 기억시킬 수 있다. 또한 정보의 가공, 순차제어 등에 이용될 수 있으며, N비트로 구성된 시프트 레지스터의 입력에 가해진 신호는 N개의 클럭이 가해진 후에야 SO에 나타나므로 클럭펄스의 주기가 t라 하면 결국 SI의 입력신호는 Nt시간 후에 SO에 나타난다. 즉, SI의 신호를 Nt만큼 지연시킨 결과가 된다. 따라서 시간지연 회로로 응용될 수 있다.

● 범용 레지스터

시프트 레지스터는 직렬자료를 병렬자료로 혹은 그 역으로 변환시키는데 사용할 수 있다. 시프트에 의하여 직렬로 입력된 정보를 시프트 레지스터의 각 플립플롭의 병렬방식으로 나타낼 수 있고, 그 시프트 레지스터에 병렬입력이 가능하도록 한다면 병렬로 입력된 자료를 시프트시켜 직렬로 출력시킬 수 있으므로 병렬자료를 직렬자료로 바꿀 수 있다.

이와 같이 여러가지 기능을 가지고 있는 시프트 레지스터를 범용 레지스터라 하며 범용 레지스터의 기능을 요약하면 다음과 같다.

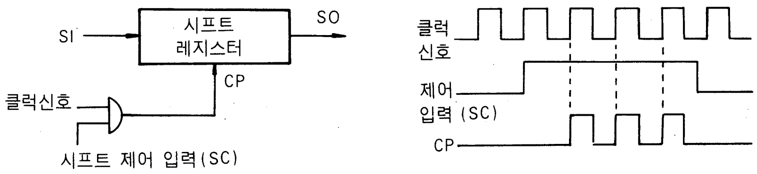
① 클리어 제어단자에 의하여 레지스터 내용을 클리어 할 수 있다.
② CP입력단자가 있어 모든 동작을 동기화시키는 클럭펄스를 입력시킬 수 있다.
③ 오른쪽·왼쪽 시프트 제어단자에 의하여 오른쪽·왼쪽 시프트를 제어할 수 있어 직렬입력과 직렬출력을 제어할 수 있다.
④ 병렬입력 제어를 함으로서 병렬전송과 N개의 입력선을 동시에 제어할 수 있다.
⑤ N개의 출력선이 있다.
⑥ 클럭펄스에 관계없이 레지스터의 정보를 보존하는 제어상태를 갖는다.

이와 같은 기능을 갖는 회로의 예를 그림 7에 나타내었다.

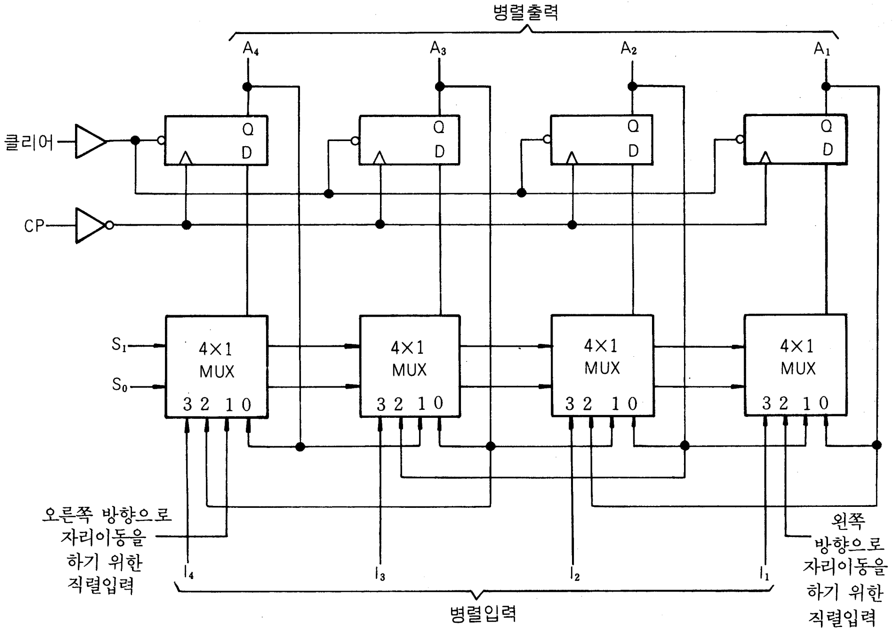
그림 7에서 4×1 MUX는 S_1, S_2입력에 의하여 표 1과 같이 연결된다.

이와 같은 범용 레지스터에 비트를 확장한 N비트 레지스터의 블록도를 그려보면 그림 8과 같다.

〔그림 6〕 시프트 제어 입력



〔그림 7〕 병렬입력이 가능한 4비트 양방향 자리이동 레지스터



〔표 1〕 그림 7의 MUX동작과 레지스터의 함수관계

S_1	S_2	FF의 D와 MUX입력	레지스터 작동
0	0	D←0	변화 없음
0	1	D←1	오른쪽 시프트
1	0	D←2	왼쪽 시프트
1	1	D←3	병렬 입력

●시프트 레지스터의 응용

디지탈 시스템에서 레지스터의 응용 범위는 대단히 넓고 다양하다. 2진수를 기본으로 취급하는 디지탈 시스템에서 2N으로 임의의 수를 나누거나 곱할 경우 임의의 수의 정보를 N비트만큼 시프트시킴으로서 나눗셈이나 곱셈 등이 처리된다.

예를 들어 십진수 13을 2진수로 나타내면 1101이고 13×4는 13×2^2이므로 1101을 2비트만큼 왼쪽으로 시프트하면 110100을 구할 수 있으며, 이것을 다시 10진수로 나타내면 52가 된다.

이와 같은 간단한 수의 계산 처리에 시프트 레지스터가 응용될 수 있음을 보여 준다. 컴퓨터나 디지탈 시스템내에서 2진 정보는 대개 병렬정보처리를 함으로서 처리속도를 빠르게 하고 있다. 그러나 장거리 정보전달을 위하여는 처리속도보다는 경제적 여건을 더 크게 고려하여야만 한다.

즉 장거리 정보전달을 병렬로 처리하기 위하여 병렬정보 비트수만큼 전송로를 구축하면, 엄청난 경제적인 비용을 감당하여야만 한다. 따라서 장거리 전송에는 직렬전송을 함으로서 속도에는 손해를 보더라도 경제적인 비용을 절감할 수 있다(그림 9 참조). 이와 같이 경우에 따라서 같은 정보라도 직렬 또는 병렬 처리방식을 선택하여야 하는데 이때 시프트 레지스터는 병렬 정보를 직렬로 혹은 그 역으로 변환시킬 수 있으므로 정보전송에 크게 유용하게 된다.

또한 시프트 레지스터는 순차제어기기 등 자동제어 시스템 등에도 유용하게 응용된다.

메모리

디지탈 시스템 특히 컴퓨터 등에서 가장 중요한 기능중 하나는 정보를 기억할 수 있는 기능이다. 전자적 스위칭 속도가 수 백만분의 1초로 빠르다하여도 그 처리능력에 맞추어 처리할 정보(혹은 자료)를 적시에 제공할 수 없으면 아무런 효과가 없다.

인간의 움직임이란 제 아무리 민첩해도 수 백분의 1초도 되기 힘들다. 따라서 전자계산기 등의 빠른 처리속도에 비하면 수만배이상 늦다고 할 수 있으며 기기들이 어떤 처리동작을 시작하기 전에 처리순서나 처리할 자료들을 미리 저장시켜 놓고 처리동작이 개시됨과 동시에 빠른 속도로 기억 내용들을 읽어내어 처리한다. 처리과정에서 다시 기억시켜 두어야 할 내용들은 역시 빠른 속도로 기억시키게 되며, 최종 결과만을 출력하도록 하였기 때문에 디지탈 시스템이 현재 각광을 받게 된 것이다.

따라서 대부분의 디지탈 시스템은 작든 크든 기억회로를 포함하고 있다. 디지탈 시스템의 대표적인 컴퓨터는 인간에 의해 인간과 비슷한 구조로 만들어졌고 좀더 인간생활의 편의를 제공하는 방향으로 발전되어 가고 있다.

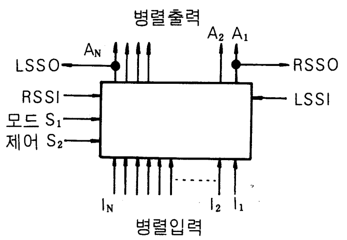
우리 인간의 신체중 가장 중요한 부분인 두뇌가 컴퓨터에서는 CPU에 대응된다. 인간의 두뇌구조에는 우선 기억할 수 있는 뇌세포가 있다. 이 뇌는 눈, 코, 입, 귀, 피부 등 소위 5감이라는 입력구조로부터 입력된 자료를 기억하는 장소이다. 이렇게 기억된 자료는 또 다른 입력자료와 비교되고 필요하다고 판단하는 형태로 변형 혹은 손, 발, 입 등의 출력부분을 통해 출력시킨다. 컴퓨터나 대다수의 디지탈시스템들 역시 입출력부분과 많은 기억장소를 갖고 있으며, 이 기억장소는 프로그램이나 자료들을 기억해두는데 중요한 역할을 한다.

●메모리의 종류

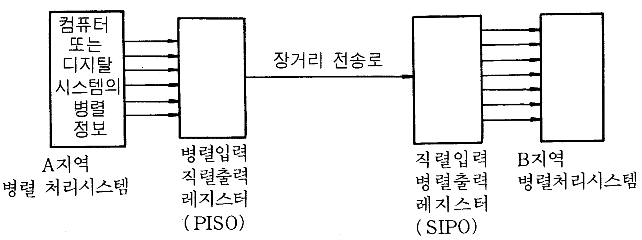
일반적으로 디지탈 시스템에 활용되는 메모리는 종류가 다양하다. 메모리의 종류를 열거하기 전에 역시 인간중심으로 기억에 관하여 생각해 보기로 하자. 우선 눈으로 혹은 다른 입력부분을 통해 들어온 자료는 뇌세포의 어느 부분에 순간적으로 기억된다. 이러한 기억상태는 자료의 성질에 따라 상당히 오랫동안 기억될 수 있으며, 보는 순간만 기억할 수 있고 잠시만 지나면 곧 잊어버리는 자료도 있다. 또 어떤 자료는 좀더 오래 보존하기 위하여 노트에 기록하여 머리에서 기억이 사라졌더라도 기록을 읽을 수 있도록 하는 경우도 있다.

이러한 예로부터 우리는 기억에 대하여 몇가지 분류를 해볼 수 있다. 즉 보고 느낄 동안만 잠시 기억할 수 있는 것, 좀더 오랫동안 기억할 수 있는 것, 노트에 연필로 기록하여 지우개로 지우기전엔 반영구적으로 기억되는 것,

〔그림 8〕 N비트 범용 레지스터의 블록도



〔그림 9〕 A와 B지역간의 장거리 정보전송

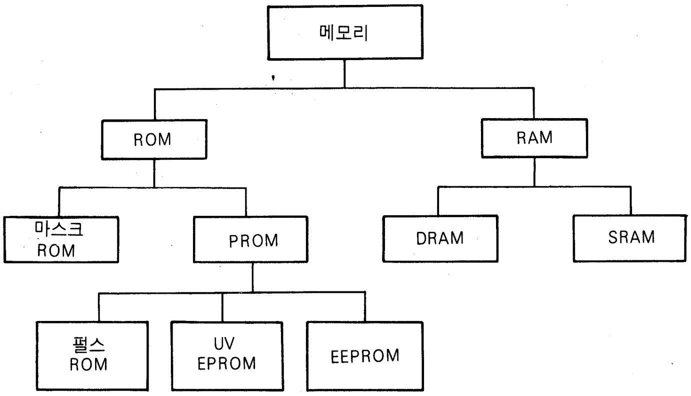


잉크를 사용하여 지울 수도 없이 영구적으로 보존되는 기억 등으로 분류하여 생각할 수 있다.

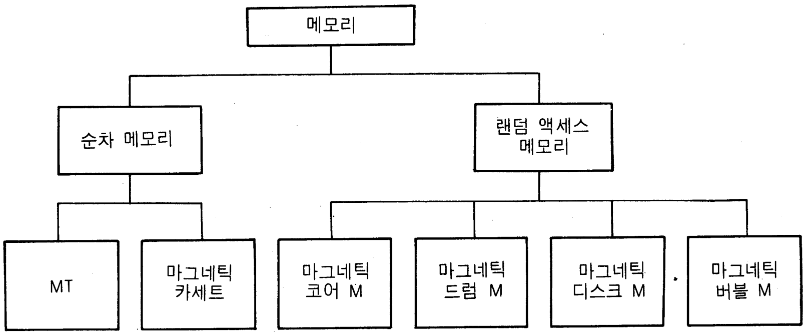
이러한 개념에서 컴퓨터의 기억장치를 분류하였을 경우에는 래치, 레지스터, RAM, EPROM, ROM 등으로 분류할 수 있다. 전자와 후자의 열거순서에 맞추어 생각해 보면 서로 상응되는 점을 알 수 있다.

메모리장치를 구성하는 형태에 따라 구분했을 때 반도체회로에 의해 구성되는 IC메모리와 자성재료를 이용하여 구성되는 자기메모리, 기타 광, 유전체 등을 이용한 기억장치들로 구분되고, 또한 메모리에 기억시키고 기억내용을 읽는 방식에 따라 순서대로 읽고 기록하는 순차메모리(Sequencial Memory)와 순서없이 자유롭게 읽고 기록하는 장소를 선택할 수 있는 랜덤 액세스 메모리(Random Access Memory)도 있다. 그림 10에 메모리의 분류관계를 나타내었다.

〔그림 10〕 IC와 마그네틱 메모리



(a) IC메모리의 분류



(b) 마그네틱 메모리의 분류

● 메모리의 구성과 동작

디지탈정보를 0이나 1의 상태로 보존시킬 수 있는 것을 정보의 기억이라 할 수 있다. 즉 어느 누가 전등 스위치를 ON시켜 두었다면 그 스위치는 ON이란 상태를 유지한다는 것은 스위치에 의하여 불이 켜질 수 있는 정보를 기억하고 있는 셈이 된다.

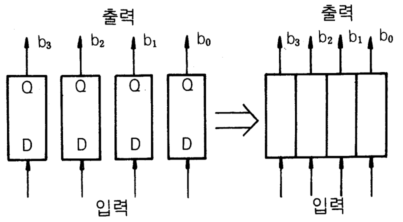
플립플롭은 입력과 클럭펄스에 의하여 출력상태가 변하며 클럭펄스를 가하지 않으면 클럭펄스가 있었을 때의 입력상태를 유지하고 있기 때문에 입력상태의 기억이 출력에 그대로 나타난다. 플립플롭 1개는 1비트 즉 0 또는 1중 하나를 기억할 수 있는 소자이다. 이 비트가 4개 모아져 있다면 4비트 기억소자가 될 것이며, 8개가 언제나 동시에 동작를 하도록 구성되어 있다면 8비트 기억소자가 될 것이다.

8비트 쌍으로 정보가 기억될 경우 각 비트의 조합으로 표현할 수 있는 양은 2^8=256개가 된다. 즉 각 비트가 00000000으로부터 11111111이 되기까지의 조합이 256가지가 되며, 각각의 조합상태에 각각 다른 의미를 부여할 경우, 예를 들어 00000000은 숫자로 나타내면 0이고, 00000001은 숫자로 나타내면 1이며, 00001111은 A숫자 등에 대응하는 표기방법을 이용한다면 0~9, A~Z 등 우리가 일상생활에서 사용하는 문자 등 256가지 이내의 정보가 표현 가능하다. 결국 8비트 조합의 한 메모리소자는 1개의 글자를 기억할 수 있는 것이 된다.

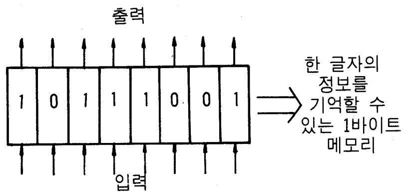
영문 한단어가 6자로 구성되었을 경우 8비트 메모리 소자 6개가 있으면 이 영문 한단어를 기억시킬 수 있다. 한 문장이 이러한 단어 100개로 이루어졌다면 8비트 기억소자 600개가 있으면 영문 한문장을 기억시킬 수 있다. 책한권이 100만자로 쓰여 졌다면 8비트 기억소자 100만개가 필요하다. 여기서 8비트로 구성된 기억소자를 1바이트 메모리라 함으로, 100만자를 기억시키기 위해서 100만 바이트의 메모리가 필요하다. 이와 같이 많은 정보를 기억시키기 위해서는 많은 양의 메모리가 필요하다. 플립플롭이 조합으로부터 여러 비트의 기억소자구성과 표시예를 그림 11에 나타내었다.

여기서 공통으로 가해지는 전원이나 클럭펄스회로 등은 생략하였다. 그림 11에서 4비트의 메모리에서는 입력이 4개, 출력이 4개로 모두 8개의 입출력선이 있다. 그리고 8비트 메모리에서는 입출력선이 16개 처리되어야 함을 알 수 있다. 따라서 이러한 메모리소자가 10개가 있다면 모두 160개의 선이, 1000개가 있다면 160000개의 입출력선이 처리되어야 한다.

〔그림 11〕 4비트 및 8비트 기억소자의 표시 예

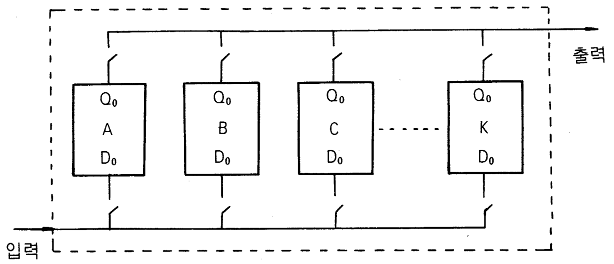


(a) 4비트 기억소자

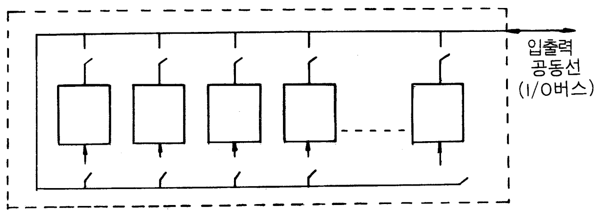


(b) 8비트 기억소자 및 기억상태 표시예

〔그림 12〕 기억소자 회로의 간소화



〔그림 13〕 입출력 공통선



이를 위한 배선을 사용자에게 처리하도록 한다면 많은 기억용량을 위한 배선은 사용자에게 있어서 거의 불가능에 가깝다. 그러나 IC 제조기술과 함께 사용자에게 입출력선 등을 쉽게 처리할 수 있는 대용량의 메모리 IC가 제작 판매되고 있으며 그 IC 내부회로의 구조 개념은 다음과 같다.

우선 1비트 메모리의 비트 수를 확장하여 생각해 보자.

먼저 **그림 12** 와 같이 1비트 기억소자 K개의 출력은 출력끼리, 입력은 입력끼리 각각 스위치를 통하여 연결시킨다. 이때 점선 네모 밖의 인입선은 입력선 1개 출력선 1개 뿐이며, 스위치를 출력에서 하나 입력에서 하나만 ON 시킨다면 선택된 기억소자의 내용만 출력될 것이다. 따라서 입력선에서 선택된 기억소자로만 입력이 가능하게 된다. 이때 1개의 입력선과 1개의 출력선을 각각 공동 입력선(Input Bus), 공동출력선(Output Bus)이라고 한다.

다시 **그림 13**과 같이 입출력선을 묶었을 경우, 메모리소자가 몇개가 되든 입출력 공동선(Input Output Bus)은 1개가 된다. 이 경우는 어느 한 기억소자에 공동선상의 정보를 기억시키고자 할 때는 그외의 모든 소자는 입력, 출력 스위치를 OFF하고 오직 기억시키고자하는 소자의 입력 스위치만 ON시켜야 한다. 출력때도 이와 마찬가지이다.

그림 14는 3개의 비트로 구성된 A, B 두개의 기억소자를 버스선으로 연결하여 간소화하는 개념도이다. 여기서 A소자의 출력 Q_0, Q_1, Q_2는 동시에 동작하는 출력스위치에 의하여 입출력 버스에 연결된다. 마찬가지로 B출력이나 A입력, B입력도 3비트가 동시에 입출력 버스에 연결되어 OFF된다. 여기서 3비트로 구성된 메모리는 3비트 메모리가 몇개로 구성되든 입출력 버스선은 구성 비트수에 대응하여 3개가 된다는 것을 알 수 있다.

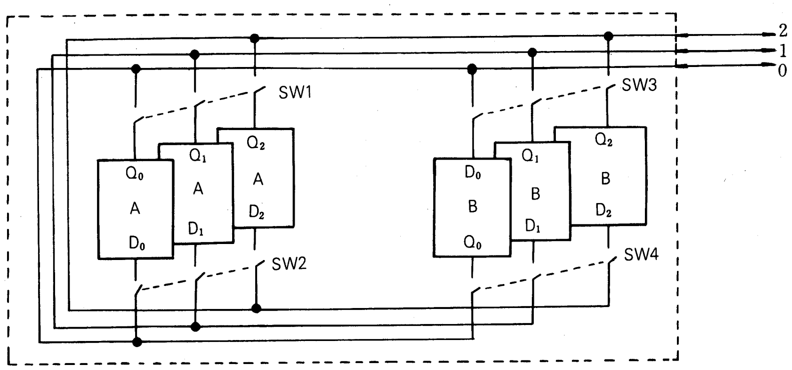
따라서 8비트 1바이트 단위의 소자로서 메모리를 구성할 경우 바이트 수의 증감에 관계없이 입출력 버스선 수는 8개가 됨을 알 수 있다.

그림 14에서 SW1이 ON일 경우 Q_0, Q_1, Q_2의 상태가 입출력선 0, 1, 2에 나타난다. 이때는 A기억소자의 상태가 출력되는 것으로 A의 기억상태를 읽었다고(Read) 표현한다. SW4가 ON 되었을 경우 입출력 공동선의 상태가 B의 D_0, D_1, D_2에 각각 입력되며 이때는 B에 기록하였다(Write)고 한다. 이 경우에 입출력 공동선을 데이터 공동선(Data Bus)이라고도 한다. 또한 이와 같이 구성되는 회로 개념의 시스템을 버스시스템이라 한다.

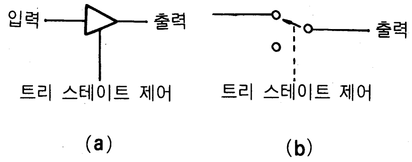
이러한 버스시스템에서 **그림 14**와 같이 사용된 기계적 스위치는 IC회로에서 트리 스테이트 게이트(Tri-State Gate : 3상태 게이트)로 대체된다. 트리 스테이트 게이트는 **그림 15(a)**와 같이 표시되며 그림 15(b)와 같은 기계적 스위치에 대응된다.

(b)의 기계적 스위치에서 트리 스테이트 제어에 의하여 출력이 입력선에 연결되면, 출력은 입력선의 상태와 같아진다. 그러나 트리 스테이트 제어에 의하여 스위치의 위치가 2의 출력선에 연결되었을 때 입력상태에 무관하게 출력선은 개방상태(이 상태를 Hi Z 혹은 High Impedance 상태)가 된다. 그림 15의 트리 스테이트 게이트는 제어단자에 논리레벨 1이 가해지면 입력이 출력에 연결된 상태이고, 0이 가해지면

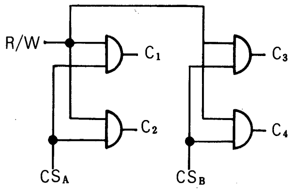
〔그림 14〕 3비트 2개의 공통선 연결



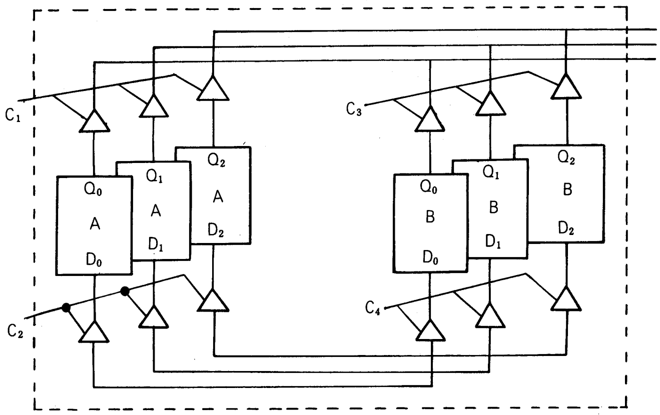
〔그림 15〕 3스테이트 게이트



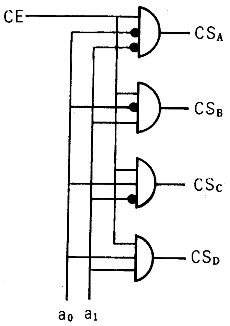
〔그림 17〕 R/W와 CS



〔그림 16〕 3스테이트 게이트를 사용한 버스회로



〔그림 18〕 어드레스와 디코더



개방상태가 된다.

다시 이러한 트리 스테이트 게이트를 그림 14와 같은 기계적스위치에 대치시켜 버스 시스템회로를 구성한 예를 그림 16에 보였다.

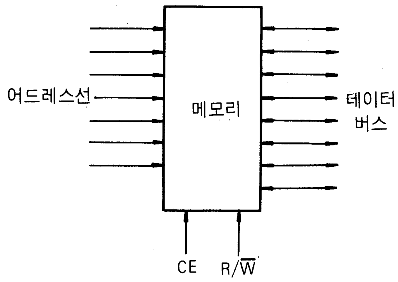
그림 16의 C_1과 C_2를 그림 17과 같이 구성하였을 경우 R/W(Read Write 선택선)와 CS(Chip 선택선)의 상태에 따라 R/W=1 $CS_A=CS_B=0$이면 $C_1=C_2=C_3=C_4=0$이므로 A, B의 입출력 모두가 개방 상태로서 버스와 무관하게 된다. R/W=1, $CS_A=1$, $CS_B=0$이면 A의 출력이 버스상에 나타난다. 즉 A기 억소자의 리드상태가 된다. R/W=0, $CS_A=0$, $CS_B=1$이면, B기억소자에 버스 상태를 라이트하는 것이 된다.

이런 방법으로 회로를 확장할 경우 버스선의 수는 기억소자의 비트 구성 숫자에 대응하고, CS선의 수는 기억소자의 갯수에 대응된다. 그러나 R/W 선은 어느 것이 확장되어도 단 한개의 선으로 가능하다.

예를 들어 1바이트로 구성되는 기억소자 100개의 회로에서는 8개의 데이터 버스선과 100개의 CS선 그리고 1개의 R/W선이 처리되어야 함을 알 수 있다. 아무리 여러개의 기억소자로 구성된 메모리일지라도 한번에 단 한개만의 기억소자가 선택되어야 하고, 또한 읽거나 쓰는 동작중 하나만 선택되어야 한다.

이러한 개념과 기억소자에 비례하여 많아지는 CS선을 간단히 처리하기 위하여 디코더(Decoder)회로를 이용한다. 예로서 CS선이 4개인 회로를 디코

〔그림 19〕 메모리의 블록도



더회로를 이용하여 구성한 회로를 그림 18에 나타내었다.

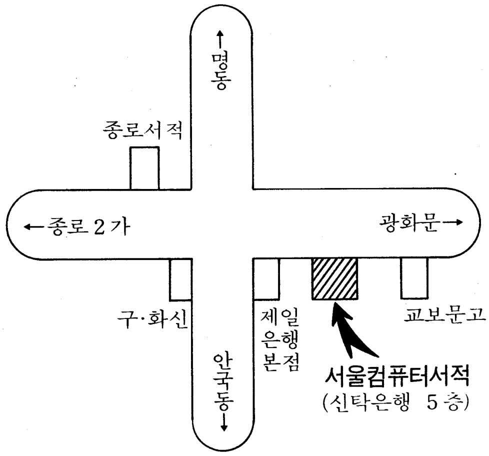
그림 18애서 볼 수 있는 바와 같이 $CS_A \sim CS_D$의 4개의 선중 a_0, a_1의 00~11까지의 조합에 의해 한개만을 1로 할 수가 있다. 즉 a_0, a_1에 의한 4선중 하나만을 선택할 수가 있다. 여기서 CE

(Chip Enable선)는 a_0, a_1이 어떤 상태이든 4개의 선중 한개도 선택되지 않거나 a_0, a_1의 어떤 상태로서 4개의 선중 1개를 선택 가능하게 할 수 있는 제어선이다. 여기서 a_0, a_1과 같은 입력선을 어드레스 선이라고 한다.

이러한 방법으로 할 경우 CS선이 1024개일 경우, 즉 기억소자가 1024개일 경우 어드레스선은 단지 10개가 필요할 뿐이다. 또한 CS선은 어드레스선의 2^n배까지 구성가능해진다.

이와 같이 구성된 메모리의 블록도는 그림 19와 같이 나타낼 수 있다. 그림 19에서 메모리 용량은 어드레스선 수의 2^n이 되며, 메모리 구성 비트수는 데이터 버스선수와 같다.

지금까지의 설명들은 플립플롭으로 구성되는 스태틱 IC 메모리 구성에 관한 예를 든 것이다. 다음호에 계속해서 기타 메모리에 관해 설명하겠다. PC



하드웨어 설계방법

김용득/아주대 전자과 교수

지난호에서 설명하였듯이 표준화된 범용 마이크로프로세서는 매우 다양하다. 따라서 먼저 어떻게 이들 다양한 마이크로프로세서를 사용목적에 따라 적절하게 선정할 수 있는 기준이 무엇인지 설명하기로 한다.

선정기준은 다음과 같은 점들을 고려하여야 한다.

첫번째로 고려할 사항은 명령어들이 얼마나 다양한가를 살피는 것이다. 여기에는 비트 처리능력, 여러 워드(Long Word)의 처리상태, 곱셈과 나눗셈 등의 연산 처리능력, 효율적인 인코딩 기능 등을 들 수 있다.

두번째로는 내부 레지스터의 활용성을 고려하는 점이다. 즉 다양한 레지스터의 수, 레지스터가 기억할 수 있는 단위정보의 폭과 특수한 목적에 쓰이는 레지스터를 어느정도 갖고 있는가를 살피는 것이다.

세번째는 주소번지 지정방식으로 사용되는 기억소자와 입·출력번지의 사용범위 및 기억소자 관리체계의 용이성을 들 수 있다.

다음으로는 외부 하드웨어 접속제어를 위한 제어버스 구성형태의 용이성, 전용 특수 프로세서(Coprocessor)의 접속 가능성과 이들 제품의 다양한 제공능력 및 참고재료의 유용성, 개발장비의 활용성, 소프트웨어의 이용성 등을 검토하여 설계목적에 따른 마이크로프로세서를 선택해야 한다.

그러나 단위 목적의 마이크로프로세서 응용시스템을 개인적으로 설계할 때 무엇보다도 중요한 것은 어느 마이크로프로세서의 제품이 주위에 가장 많이 보급되어 있으며, 쉽게 구입하여 활용할 수 있는지의 여부에 따라 선정해야 한다. 우리나라 현실에서 대부분의 16비트 마이크로프로세서는 인텔계열의 마이크로프로세서인 8086, 8088, 80286 등이 사용되고 있으므로 이를 기본으로한 하드웨어 설계방법을 설명하기로 한다.

범용 마이크로프로세서의 특성

먼저 IBM-PC/XT에 사용되고 있는 마이크로프로세서인 8086, 8088의 특징을 비교해 보자. 8086은 진정한 16비트 마이크로프로세서로 16비트의 외부정보선을 갖고 있어 한번에 기억장치로부터 16비트씩 정보호출이 가능하다. 반면에 8088은 8비트와 16비트 혼합형이며, 마이크로프로세서 내부에서는 16비트로 처리되지만 외부신호와 정보전송에는 8비트로 이루어진다. 그러나 이들 두개의 마이크로프로세서는 소프트웨어적으로 호환성이 있어 함께 사용할 수 있다. 특히 이들 마이크로프로세서는 수치계산을 위한 8087 마이크로프로세서와 입·출력전담 마이크로프로세서인 8089와 함께 구성되어 시스템 효율을 증가시킬 수 있다.

IBM-PC에서는 주로 하나의 마이크로프로세서로 구성한 최소모드를 사용하나, 여러 마이크로프로세서로 구성된 다중 마이크로프로세서 시스템(최대모드 구성방식)이 가능하며, 1MB(10^6 바이트) 주기억장치를 직접활용 가능하다.

80286에서는 8086/88과는 달리 여러개의 분리된 독립 프로그램을 동시에 처리할 수 있는, 즉 실시간 다중업무 시스템구성이 가능하다. 또한 다중 마이크로프로세서 구성을 위한 수치계산용 주변소자 제어기, 이더네트(Ethernet) 등과 같은 PC통신을 위한 많은 보조용 집적회로가 제공되고 있다. 또한 80286에서는 1GB(10^9 바이트)의 가상메모리 구성방식이 가능하며, 비록 마이크로프로세서라고는 부르지만 대형 컴퓨터의 역할과 동일하다고 볼 수 있다.

참고로 80386이란 32비트 마이크로프로세서로 80286과 똑같은 내부구성 및 운영되는 소프트웨어 기술이 적용되고 확장된 마이크로프로세서이다.

그림 1의 8086 내부회로 구성도에서 이들 16비트 마이크로프로세서는 파이프라인(Pipeline)방식으로 작동됨을 알 수 있다. 파이프라인 방식이란 내부 CPU회로 구조가 그림 1과 같이 2개의 처리부, 즉 명령어 처리부(A버스)와 명령어 읽음 동작부(B, C 버스)의 2개부가 독립적으로 구성되어져 있어 동시에 작동가능함을 의미한다. B버스와 C버스가 연결된 명령어 처리부는 기억소자로부터 명령어 코드를 읽어 대기 레지스터에 저장하는데 사용되며, A버스에 연결된 정보처리 기능과는 별도로 그림 2와 같이 동시 운영이 가능하다.

그림 1에서 8086 마이크로프로세서에서 부착가능한 기억장치 영역은 1MB이며, 이는 20비트로 구성되어야 한다. 그러나 내부구성이 16비트이므로 이들 주소 정보를 처리하기 위하여는 16비트 단위로 구성되어야 한다. 이러한 16비트 단위 즉 64KB로 블록을 나누어 운영하게 되며 이를 세그먼트라 정의하고, 다음 호의 소프트웨어에서 내부 레지스터와 함께 다시 설명하기로 하겠다.

그림 1의 명령어 대기 레지스터는 프로그램이 저장된 기억소자에서 하나의 명령어를 읽어 이를 처리하도록 되어 있으며, 이들 동작 처리시간을 단축하기 위하여 그림 2의 (b)에서와 같이 읽음동작과 처리동작을 동시에 처리하는 이른바 파이프라인 방식을 사용한다. 범용 레지스터는 정보처리를 위하여 사용되는 레지스터로 볼 수 있다.

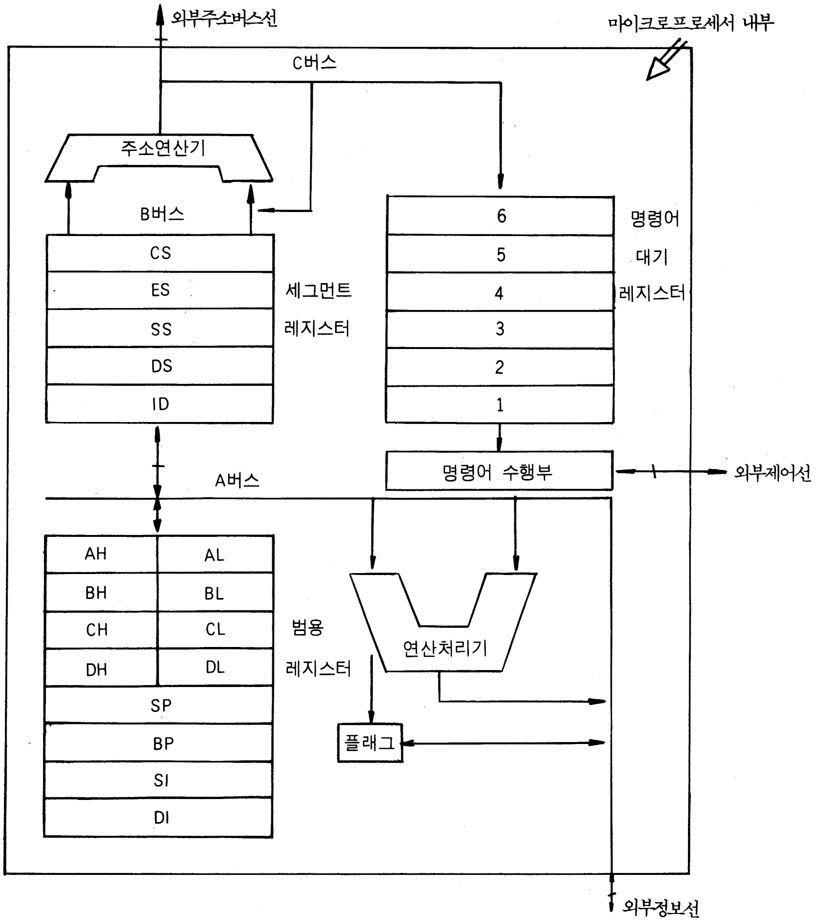
하드웨어 시스템의 설계

하드웨어 설계에 있어 그림 1에서처럼 내부 버스선에 연결된 외부단자의 기능이 어떠한 것이며, 또한 어떤 특성을 갖는가를 이해해야 한다.

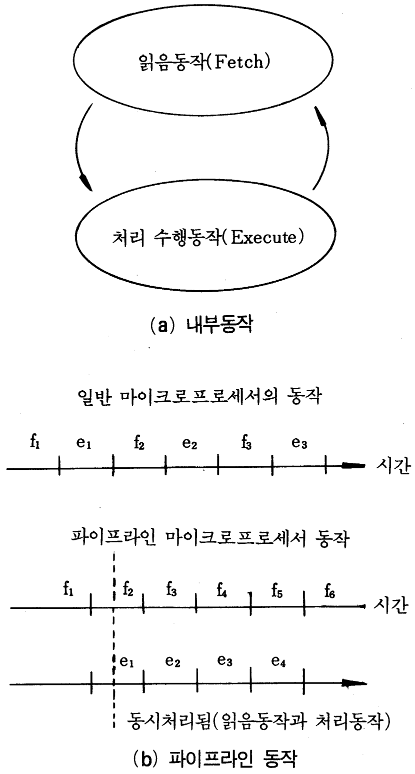
그림 3의 마이크로프로세서 외부단자에서 8086/88 마이크로프로세서(설명을 쉽게하기 위하여 8086만 고려하도록 하나 8088/80286 모두 유사하다.)는 가능한한 다양한 응용에 적용하기 위하여 두개의 동작모드, 즉 최소구성모드와 최대 구성모드를 갖으며 33번 단자(MN/$\overline{MX}$)로 결정한다. 이 단자가 “1”이면 최소모드, “0”이면 최대모드로서 시스템에 사용되는 프로세서의 갯수를 1개 또는 2개 이상임을 지정하고 있다.

AD0부터 AD15의 16개 단자는 주소번지와 정보신호들을 다중송신하고

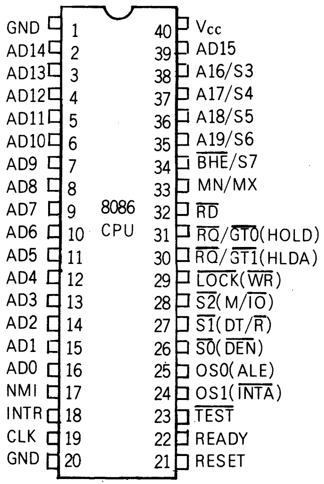
〔그림 1〕 8086 내부회로 구성도



〔그림 2〕 8086 마이크로프로세서의 동작



〔그림 3〕 마이크로프로세서의 외부단자



상위 4개 주소번지와 함께 20개 주소 방식을 지정한다. 이때, AD선은 ALE (25번) 단자에 의하여 "1"이면 주소선을, "0"이면 정보선을 표시하도록 한다.

즉, 이는 단일단자로 지정할 경우 주소선 20개, 정보선 16개인 총 36단자가 요구되지만, 8086 방식에서는 21개선으로 모두 지정이 가능하게 되어 있어 외부 단자수를 40단자 이내로 구성이 가능하게 하였다.

또한, 34번 단자인 $\overline{BHE}$는 바이트 전송을 할 것인지, 또는 워드전송을 할 것인가를 지정하고 있다. 17(NMI), 18 (INTR) 단자는 외부 하드웨어에 의하여 긴급히 처리할 특정 프로그램에 사용되는 인터럽트를 요구하기 위한 단자이고, 24($\overline{INTA}$)는 프로세서의 응답을 알리는 단자이며, 19(CLK)단자는 마이크로프로세서 안에서 동작을 동기화시키는 클럭신호를 제공하기 위한 것이다.

21(RESET)단자는 마이크로프로세서 시스템을 처음부터 다시 작동시키기 위하여 사용자의 스위치 조작으로 이루어지며, 항상 "FFFF0"번지의 프로그램부터 작동한다.

따라서 이 주소번지는 시스템을 초기화하고, 응용 소프트웨어나 운영체제를 적재하기 위한 프로그램이 들어있는 ROM이 할당 되어야만 한다. 그리고, 이들 프로그램을 부트스트랩 로더(Bootstrap Loader)라 부른다.

22번(READY) 단자는 저속 주변소자와 프로세서의 동작시간을 일치하기 위하여 사용되며, 23번(TEST) 단자는 WAIT 명령과 결합되어 다중 프로세서 시스템에서 사용한다.

또한, 입·출력 장치는 주소지정 방식과 포트지정 방식으로 접속가능하며, 이는 M/$\overline{IO}$(28번) 단자와 $\overline{RD}$(32), $\overline{WR}$(29)단자가 함께 지정된다.

이들 단자는 앞으로 입·출력 장치의 접속회로편에서 자세히 설명하도록 하겠다.

최소모드에 의한 하드웨어 구성

8086 마이크로프로세서의 외부 접속단자가 이해되면, 이를 사용한 기억소자의 입·출력 접속소자와 같은 LSI소자들을 접속할 수 있는데, 이들 접속방식에 관한 연구를 하드웨어 설계라 부른다.

16비트 프로세서를 사용한 하드웨어 설계방식에는 최소 구성모드와 최대 구성모드로 설계 가능하며, 앞에서도 언급했지만 이는 MN/$\overline{MX}$ 단자에 의하여 결정된다. 즉 최소모드 구성을 위하여는 이 단자에 5V를 지정하면 된다(그림 4 참조).

그림 4의 최소모드 시스템회로에서 ALE 단자가 "1" 상태일 때 동기되어 AD 버스선상에는 주소번지가 나타난다.

또한, 이 신호를 사용하여 8282 래치회로를 작동하여 AD선상의 값을 기억시키며, 새로운 주소번지에 지정하게 된다. 정보는 2개의 8286 트랜시버를 통하여 정보선에 접속하고, 제어선은 직접 주변소자에 연결하도록 한다.

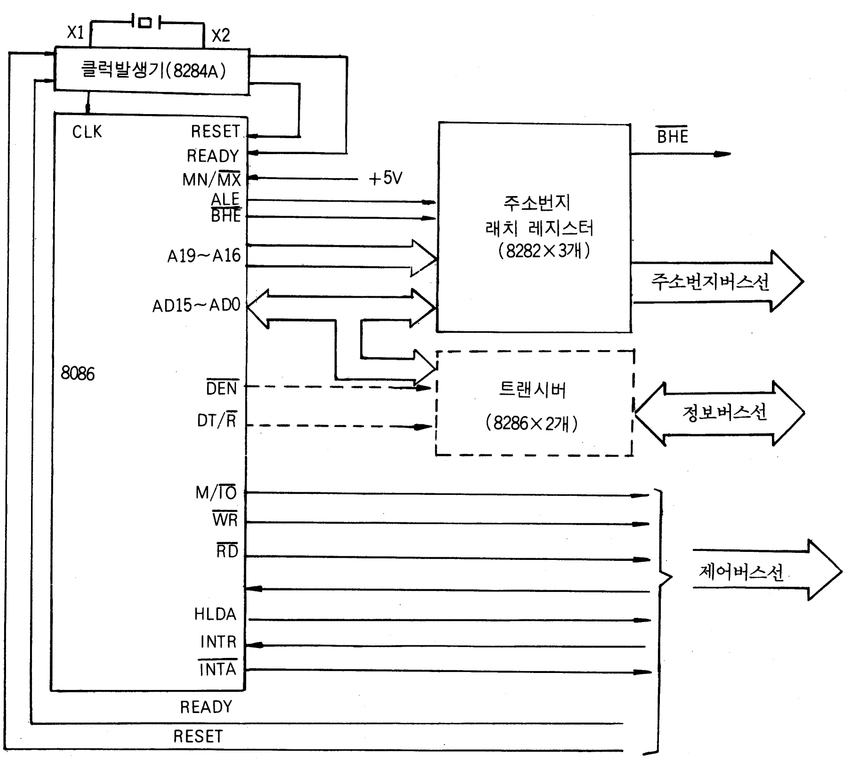
여기서 8282는 8비트 래치이기 때문에 16비트 주소번지를 필요할 때는 2개, 전체 20비트 주소번지가 사용되면 3개가 필요하게 된다.

또한 8286 트랜시버는 8개의 3상태 입·출력을 가지므로 8086 마이크로프로세서에서는 2개가 그림 4와 같이 구성된다.

클럭발생기인 8284는 X1, X2 단자에 연결된 수정발진자에 의하여 클럭이 입력주파수의 1/3배인 클럭이 발생된다.

이는 전 시스템 동기와 구동을 위하여 사용되고, 이를 시스템클럭이라 부르고 외부 접속회로 설계 및 응용편에서 어떻게 활용되는가를 시간계산 예로서 설명하겠다.

〔그림 4〕 최소모드 시스템



최대모드에 의한 하드웨어 구성

MN / $\overline{MX}$ 단자가 접지되어 있는 경우에는 최대모드로 작동한다. 최소모드와의 차이점은 제어신호를 해석할 부가적인 회로의 필요성에 의하여, S2, S1, S0 상태비트에 의하여 정보전송 제어기의 상태를 나타내는 점이다. 즉 S0=S1=1이고 S2가 "0"이면 주변 접속회로와 마이크로프로세서간의 정보 전송임을 의미하고, S0=S1=S2=1이면 접속소자와 기억소자간의 전송임을 의미한다(그림 5 참조).

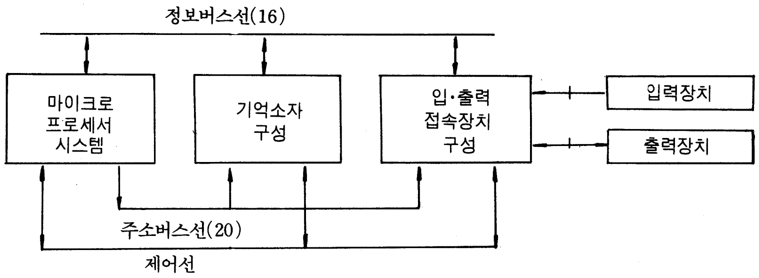
여기서 LOCK단자는 다중 마이크로 프로세서 시스템 설계시에만 사용되는 것으로 버스의 사용상태를 표시한다.

그림 5에는 $\overline{S0}$, $\overline{S1}$, $\overline{S2}$ 단자들이 마이크로프로세서로부터 대응되는 상태를 받아들여 $\overline{INTA}$, $\overline{IORC}$, $\overline{IOWC}$, $\overline{WRDC}$, $\overline{MWTC}$ 신호를 발생시킨다. 이때 ALE, DT / $\overline{R}$, DEN 단자는 최소모드에서의 정의와 같다. $\overline{AEN}$, IOB, CEN 단자는 다중 마이크로프로세서 시스템으로 사용하기 위한 것으로, 만약 마이크로프로세서가 1개만 사용되는 시스템에서는 AEN과 IOB는 접지시켜 놓고 CEN=1을 인가한다.

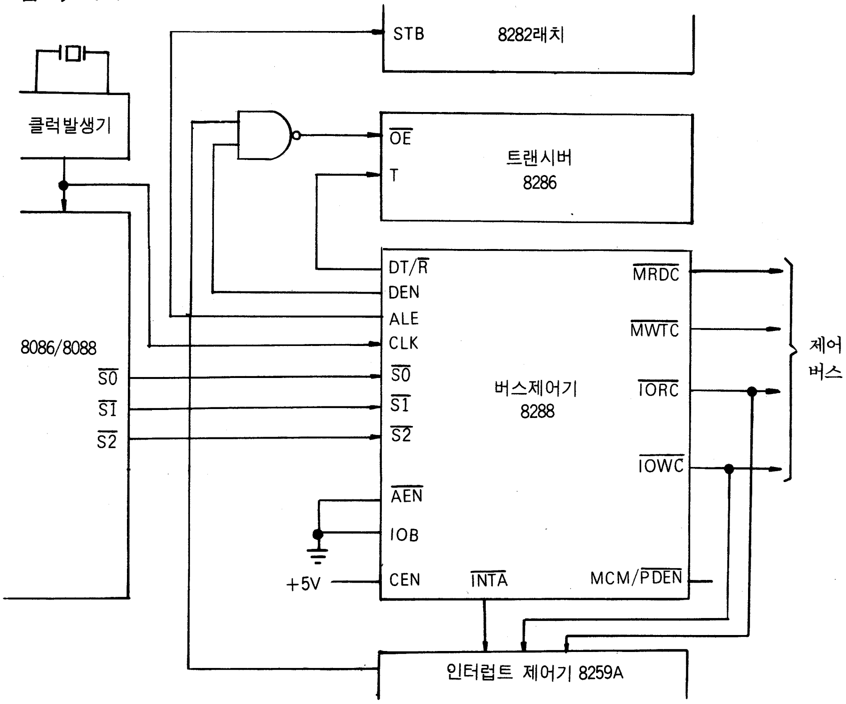
그림 5의 출력기능에서 $\overline{INTA}$ 단자는 S0=S1=S2=0 일 때, 두개의 인터럽트 알림펄스를 우선순위로 갖는 인터럽트 제어기나 인터럽트 장치에 알려주며, $\overline{IORC}$는 입 · 출력장치에 정보버스선에서 지정된 포트에 포함된 정보를 넣도록 명령하고, $\overline{IOWC}$는 입 · 출력 접속소자에 정보를 받아들여 지정된 포트에 출력하라는 명령이다. $\overline{MRDC}$ 와 $\overline{MWTC}$는 기억장소로부터 내용값을 읽어들이거나, 마이크로프로세서로부터 기억소자에 기억시키라는 명령을 수행하는 신호단자로 "0" 상태에서 작동된다.

상기와 같은 마이크로프로세서 시스템의 구성은 CPU상태 구성을 위한 설계방식이고, 실제 사용 가능한 시스템은 그림 6과 같이 기억소자와 입 · 출력 접속장치가 접속되어야만 한다. 또한 이들 모든 설계방법이 고려되어야만 마이크로프로세서 시스템이 설계될 수 있다고 볼 수 있다. 따라서 입 · 출력 접속장치부는 5, 6부에서 자세히 논하기로 하고, 본 호에서는 기억소자 시스템 설계에 관해서만 추가로 설명한다.

(그림 6) 전체 시스템의 블록 구성도



(그림 5) 최대모드의 구성



기억소자 시스템의 설계

업무의 처리순서를 마이크로프로세서가 이해하는 언어로 작성한 프로그램과 처리될 정보는 기억소자에 저장되어져야 한다. 따라서 본절에서는 이들 제어소자구성과 마이크로프로세서 시스템과의 접속방식에 관하여 기술하겠다.

기억소자는 정보를 어떻게 읽고, 저장하느냐의 방식에 따라 주소 호출방식(RAM : Random Access Memory)과 순서대로 읽고 쓰게되는 순서 호출방식(SAM : Sequential Access Memory) 으로 나누어진다.

그러나 마이크로프로세서 시스템에서는 모두 주소 호출방식의 기억소자가 사용되어고 있으므로, 이 방식의 하드웨어구성과 접속방식에 대하여 설명하겠다.

주소 호출방식의 LSI화된 기억소자는 크게 전원이 인가된 경우에 임의의 주소에 정보를 저장할 수 있는 RAM(읽고 쓸 수 있는 소자)과 전원 인가상태와 무관하게 항상 일정한 정보를 갖고 있는 ROM(읽을 수만 있는 소자)으로 구분되며, 내부 구성형태는 **그림 7**과 같다.

여기서 $\overline{CS}$(Chip Select)는 기억소자를 사용하고자 할 때 "0"의 신호를 인가해야 한다. 읽고($\overline{OE}$), 쓰기($\overline{WE}$) 위하여 해당되는 단자에 "0"의 신호를 인가하여 주소번지에서 지정한 위치의 내용을 읽고 쓸 수 있다.

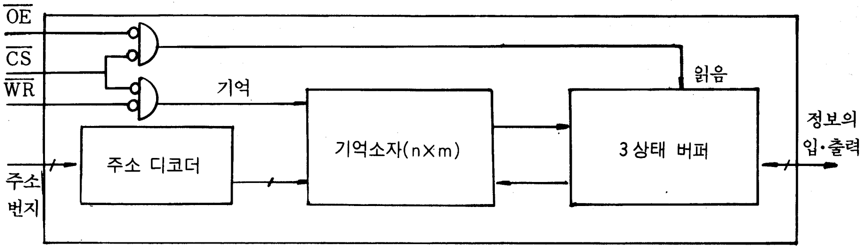
상품화된 IC는 저장용량이 다양하고 (1KB~4MB), 출력비트 형태도 또한 다양하므로 (1, 4, 8 비트) 이들 단위소자를 사용하여 필요한 기억소자 시스템을 구성해야 한다.

기억소자 시스템 설계방식은 출력비트를 확장하는 워드확장법(Word Expansion)과 용량을 확장하기 위한 주소확장법(Address Expansion)이 있다. 워드 확장방법은 필요한 출력비트가 되도록 단위소자를 병렬로 배열한 후, 모든 주소단자와 제어단자를 묶어 사용하는 방법이고, 주소 확장방법은 용량을 확장하기 위한 방법으로서 해당소자를 주소영역표(Memory Map)라 부르는 특정번지에 할당하여, **그림 8**과 같이 구성한다.

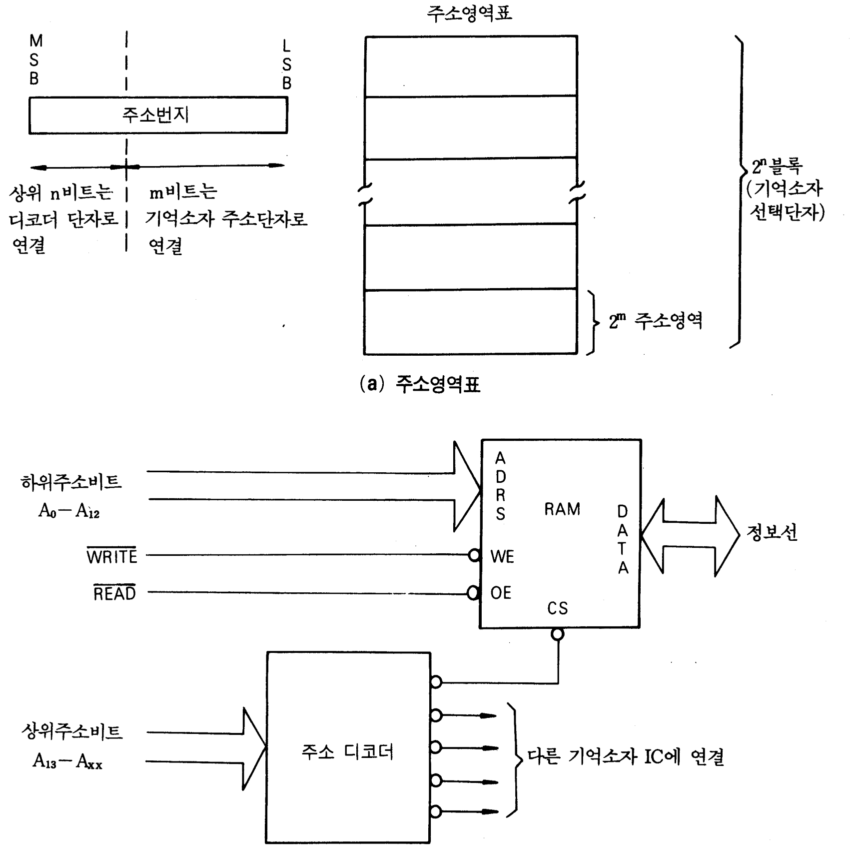
예를 들어 그림 8(a)와 같이 주소영역을 사용하고자 하면, 기억소자 시스템 구성은 그림 8(b)와 같이 된다. 이때 기억소자 시스템 단자에 나와있는 주소번지(A0~A19)와 제어신호(RD, WR) 및 정보선(D0~D15)은 직접 앞절에서 설계한 마이크로프로세서 시스템에 접속시키면 된다.

즉 마이크로프로세서로부터의 상위 주소와 제어신호를 단위기억소자의 제어신호에 연결시키는 회로구성방식을 기억소자의 접속회로라 부른다.

〔그림 7〕 기억소자 IC의 내부구성

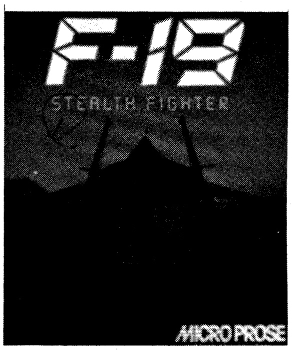


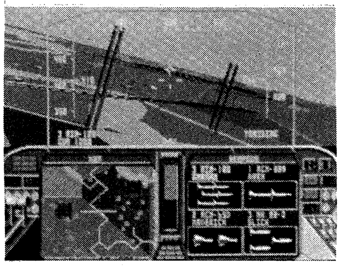
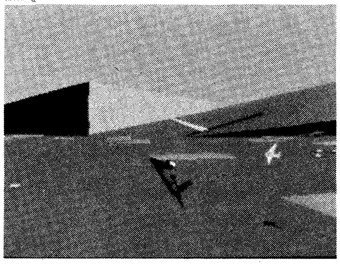
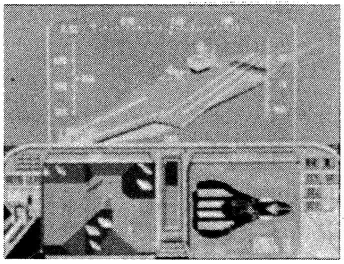
〔그림 8〕 기억소자 시스템의 접속장치 구성



(a) 주소영역표

(b) 접속회로



NAME	SIDE
10TH FRAME PROBOWLING	1
2 ON 2(BASKET BALL)	1
221B BAKER STREET	2
3D HELICOPTER	1
4TH AND INCHES	1
ACE OF ACES	1
ALTERNATE REALITY - THE CITY	2
ANCIENT ART OF WAR AT SEA	1
ARTIC FOX	1
ATLAS	1
BAD STREET BRAWER	1
BALANCE OF POWER	3
BARD'S TALE I	2
BATTLE HAWKS 1942	2
BATTLE ZONE	1
BIONIC COMMANDO	1
BLACK CAULDRON	2
BLACK JACK	1
BLACK JACK ACADEMY	1
BLOCK BUSTER	1
BOP,N WRESTLE	1
BREACH	1
BUBBLE GHOST	1
BUCK ROGERS	1
BY FIRE OF SWORD	1
CALIFONIA GAME	1
CANFIELD CARD GAME	1
CHAMPIONSHIP BOXING	1
CHAMPIONSHIP GOLF	1
CHUCK YEAGER'S FLIGHT TRAINNER	1
COMMANDO	1
DALLAS QUEST	1
DAM BUSTER	1
DARK CASTLE	2
DEATH SWORD	1
DEFENDER OF THE CROWN	1
DESTROYER	1
DRAGON WORLD	2
EAGLE	1
EARL WEAVER BASE BALL	2
F-15 STRIKE EAGLE	1
F-19 STEALTH FIGHTER	3
FALCON F-16	1
FAMILY FUED	1
FIRE POWER	2
FIRST EXPEDITION	2
FLIGHT SIMULATOR	1
FLIGHT SIMULATOR V3.0	2
FORD SIMULATOR	1
GAMESTAR BASEBALL	1

NAME	SIDE
GATO	1
GAUNTLET	1
GHOSTS'N GOBLINS	1
GRANDPRIX CIRCUIT	2
GUNSHIP	2
HACKER II	1
HARD BALL	1
HARRIER	2
HEROES OF THE LANE	3
HUNT FOR RED OCTOBER	1
IKARI WARRIOR	1
IMAGIC TENNIS	1
INDOOR SPORTS	2
INFILTRATOR	1
INFOCOMIC	2
INTO THE EAGLE'S NEST	1
JACK NICKIAUS GOLF	2
JEOPARDY	1
JET	1
KARNOV	2
KING'S QUEST I,II,III,IV	1
KNIGHT ORC	2
LAST NINJA	2
LEISURE SUIT LARRY II	6
LODE RUNNER	1
MAD SPY VS SPY III	1
MAN HUNTER	5
MARBLE MADNESS	2
MASTER NINJA	1
MIGHT & MAGIC	2
MIXED UP MOTHER GOOSE	2
MOEBIUS	2
NEW SOKO BAN	1
NFL CHALLENGE	2
NIGHT MISSION PINBALL	1
NINJA	1
OFF SHORE WARRIOR	1
PATTON VS ROMMEL	1
PETE ROSE PENANT FEVER	2
PHANTASIE I,III	1
PHYCHO	2
PINBALL CONSTRUCTION SET	1
PINBALL WIZARD	1
PIRATES	2
PIT STOP II	1
POLICE QUEST I,II	3
POOL OF RADIANCE	3
PROLEAGE SPORTS	1
PT-109	1
QUSTRON II	2

NAME	SIDE
RAD WARRIOR	1
RAMPAGE	1
RINGS OF ZILFIN	2
ROADWAR 2000	1
SCRABBLE	1
SDI	1
SEARCH AND DESTROY	1
SEVEN CITIES OF GOLD	1
SHANG HAI	1
SHOGUN	1
SKATE OR DIE	2
SKY RUNNER	1
SLIENT SERVICE	1
SPACE QUEST I,II	2
SPIDER MAN	1
STAR COMMAND	3
STAR FLEET I	1
STAR FLIGHT	2
STAR GLIDER	1
STRIP BLACK JACK	1
STRIP POKER	1
SUB BATTLE	1
SUMMER GAME II	1
SUPER STAR ICE HOCKEY	1
SUPERBYKE	1
TEED OFF	1
TEST DRIVE	1
THE GAMES(SUMMER EDITION)	4
THE LAND OF THE LOUNGE LIZARDS	2
THEXDER	2
THREE STOOGES REEL II	2
TOP GUN	1
TREASURE ISLAND	2
TROJAN	1
TWILIGHT'S RANSOM	2
ULTIMA I,II,III,IV,V	1
WHERE IN THE EUROPE IS CARMEN-	1
WHERE IN THE USA IS CARMENSAN	1
WHERE IN THE WOLRD IS CARMEN	1
WINTER GANE	1
WIZARD WARS	2
WIZARDRY IV	1
WORLD GAME	1
WORLD KARATE CHAMPIONSHIP	1
WORLD TOUR GOLF	1
DOUBLE DRAGON	2
CONTRA	1
ROMANCE OF THE THREE KINGDOMS	3
SKYFOX 2	1

간단한 방법으로 키 정의를 바꾼다

컴퓨터의 키보드에서 이미 정의되어 있는 값을 자기 마음대로 바꾸거나 어떤 키 하나를 누르면 원하는 일련의 문장을 나타나게 하면 어떨까?

컴퓨터에 점차로 심취해 가고 있는 독자라면 이 말만 듣고도 벌써 귀가 솔깃해 있을 것이다. 쉬운 예로 실수의 빈도가 많은 콜론(:)과 세미콜론(;) 키를 분리해 놓는다거나 리턴키를 편리한 위치로 옮겨 버리는 경우를 들 수 있겠다. 더 나아가 DOS상태에서 자주 쓰이는 DIR, TYPE, COPY 등의 명령문구를 단 하나의 키를 눌러 화면에 출력시킬 수도 있다.

이런 식으로, DOS상태에서 서로 다른 기능키를 최고 40개까지 정의하여 각종 명령어 문구들을 대치시킬 수 있는데 이러다가 자기가 갖고 있는 키보드가 전혀 다른 타입으로 변해버려 키의 지정이 엉망으로 되지나 않을까 하는 독자가 있다면 우선 안심하라고 말하고 싶다. DOS디스켓에는 이미 표준형 키보드 드라이버가 기억되어 있어서 언제든지 시스템을 재부팅시키면 원래의 기능을 얻게 된다. 키의 재정의는 퍼스널컴퓨터 89년 2월호(115p)에 소개된 바 있으나 이번에는 같은 원리를 가지고 베이직 이용자들을 위해 설명하고자 한다.

CONFIG.SYS의 부팅

처음에 해야할 작업은 CONFIG.SYS화일을 만드는 것이다. 이 작업을 하는 이유는 컴퓨터가 부팅될 때 키보드를 제어하는 디바이스 드라이버를 인스톨시키기 위한 것으로 DOS 디스켓에 있는 라인에디터(EDLIN.COM)을 이용한다. DOS 프롬프트 상태에서 EDLIN CONFIG.SYS라고 입력한 후 〈ENTER〉키를 누르면 화면에 다음의 메시지가 나온다.

```
New file
*
```

계속하여 다음과 같이 입력한다.

```
1i
DEVICE=ANSI.SYS
```

입력이 끝났으면 〈Ctrl〉-〈Break〉와 'E'를 눌러 기억시키면 되는데 간단히 DOS상태에서 COPY CON CONFIG.SYS를 입력하는 방법도 있다.

```
A:\>COPY CON CONFIG.SYS
DEVICE=ANSI.SYS
^Z
   1 file(s) copied
```

이렇게 하여 만들어진 CONFIG.SYS는 컴퓨터가 부팅될 때 자동적으로 실행되며 컴퓨터는 어떤 새로운 키의 할당을 받아들일 준비상태가 된다.

키정의의 변환

다음은 키를 실제로 변환하는 단계이다. 베이직을 이용하여 필요한 데이터 값을 텍스트 화일로 만들어 디스켓에 저장한다. 리스트 1은 이런 테크닉의 한 예를 보인 경우이다. 여기서는 대문자인 'Q'를 'D'로 변환시키는 것이다. 리스트 1을 REASSIGN.BAS란 이름으로 저장한 후 실행시키면 다음의 문자열을 갖는 텍스트 화일이 만들어진다.

```
ESC["Q";"D"p
```

CHR$(27)은 ESC 캐릭터를 나타내는 ASCII코드로 여기서는 대문자인 "Q"를 "D"로 바꾸는데 필요한 제어코드로 쓰이고 있다. 한가지 잊지 말아야 할 점은 재부팅되는 디스켓 속에 조금전에 작성한 CONFIG.SYS이 있어야 한다는 것이다.

〔리스트 1〕 REASSIGN.BAS

```
10 A$=CHR$(27)+"["+CHR$(34)+"Q"+CHR$(34)+";"+CHR$(34)+
   "D"+CHR$(34)+"p"
20 OPEN "KEY.TXT" FOR OUTPUT AS #1
30 PRINT #1,A$
40 CLOSE #1
```

재부팅되면서 나오는 시간과 날짜에 적절히 응답한 후 DOS 프롬프트상태에서 TYPE KEY.TXT를 입력하여 〈ENTER〉 키를 누른다. 이때 특수한 제어캐릭터들이 컴퓨터의 메모리 속으로 들어가게 되는 것이다. 이제부터는 키보드에서 대문자 'Q'키를 누르면 컴퓨터는 이것을 "D"로 받아들여 화면에 출력시키게 된다.

키보드 매크로 정의

위의 기법을 조금만 응용하면 키보드 매크로 즉, 키보드에서 하나의 키를 누르면 여러개의 문자 또는 문구를 출력해 주는 기능을 얻을 수 있다. 이해를 돕기 위해 이번에는 대문자 'Q'를 누르면 'The Message'가 출력되도록 해보자. 위에서 작성한 REASSIGN.BAS프로그램을 다시 로드하고 10번 문장을 다음과 같이 고쳐 보자.

```
10 A$=CHR$(27)+"["+CHR$(34)+"Q"+CHR$(34)+";"+CHR$(34)+
   "The Message"+CHR$(34)+"p"
```

이 프로그램을 다시 실행시키면 이번에는 다음의 캐릭터를 담은 텍스트화일이 만들어 질 것이다.

```
ESC["Q";"The Message"p
```

SYSTEM을 입력해 DOS로 빠져나간 다음 다시 TYPE KEY.TXT라고 입력한다. 이제부터는 'Q'를 누르면 'The Message'가 출력된다. 그리고 KEY.TXT화일이 만들어 질 때 캐릭터 대신에 ASCII코드를 사용해도 무방하다. 예를 들어 앞에서 'Q'를 'D'로 변환하는 경우에 10번 문장을 다음과 같이 표현해도 동일한 효과를 얻게 된다.(여기서 68은 D의 ASCII코드이다.)

```
10 A$=CHR$(27)+"[81;68p"
```

40개까지 가능한 기능키

이제부터는 좀 더 특별한 기능키에 대해서 알아 보자. 확장 ASCII코드를 제공함으로써 10개의 함수키 또는 〈Ctrl〉, (Alt〉, 〈Shift〉키와의 조합에 의한 함수키들을 재정의할 수 있다.

이러한 키의 조합은 4개가 모일수도 있고, 10개 또는 최고 40개의 키가 모여 이룰 수도 있다.

이번에는 REASSIGN.BAS의 10번 문장을 다음과 같이 고쳐서 실행해 보자.

```
10 A$=CHR$(27)+"[0;84;"+CHR$(34)+"DIR"+CHR$(34)+";"13p
```

이 경우에 만들어지는 텍스트 화일은 다음의 내용을 가지며 84앞에 있는 0가 컴퓨터에게 확장 키코드(특수한 키의 조합 신호를 지님)를 찾게 한다. 예를 들어 확장 키코드 84는 〈Shift〉-〈F1〉을 나타낸다. 우리는 방금 이 확장 키코드를 'DIR'과 캐리지 리턴코드가 연결되게 재정의했다. 마찬가지 방법으로 프로그램을 실행시킨 후 DOS로 나온 다음 TYPE KEY.TXT를 실행한다. 이제 〈Shift〉와 〈F1〉키를 누르면 디스크의 디렉토리가 출력된다.

정의된 캐릭터 순서에서 'p'바로 전에 위치한 숫자 13은 〈ENTER〉를 나타내는 ASCII코드로 키보드에서 직접 〈ENTER〉 키를 누르는 것과 똑같은 효과를 얻을 수 있다.

화면모드와 컬러지정

화면의 컬러를 변경하거나 화면 해상도 등을 변환시킬 때도 같은 방법을 쓸 수 있는데 컬러를 바꿀 때는 10번 문장에 있는 소문자 'p'를 'm'으로 고쳐쓰고 컬러코드 번호를 입력하면 된다. 다음과 같이 10번 문장을 수정하여 실행시킨 다음 새로운 KEY.TXT화일을 만들어 보기 바란다.

```
10 A$=CHR$(27)+"[37;44m"
```

이번에는 KEY.TXT의 내용이 ESC[37;44m으로 되고 DOS상에서 TYPE(d)되면 화면이 파란색으로 변하게 된다. 또한 화면모드는 10번 문장을 다음과 같이 수정하면 된다.(전 예에서 'm'대신에 'h'가 쓰이고 있음)

```
10 A$=CHR$(27)+"[=1h"
```

이 경우는 화면을 40✻25 컬러 텍스트모드 상태로 만들어 준다. 만약 320✻200 컬러 그래픽모드를 원한다면 10번 문장의 숫자 1을 4로 바꾸어 주기만 하면 간단히 해결된다. 기타 자세한 화면모드와 컬러코드는 DOS매뉴얼을 참조하기 바라며 시스템을 재부팅시키면 다시 원래의 키정의로 환원된다는 것을 알아 두자.

dBASE Ⅳ의 새로

송상섭 /금산 위성통신지구국

> 아쉬튼 테이트사는 데이터베이스 패키지로 많은 PC 사용자들로부터 사랑을 받아 왔으며, 그에 보답이라도 하듯이 dBASE Ⅳ를 내놓고 사용자들의 새로운 관심을 끌고 있다.
> 이미 dBASE Ⅲ+에 익숙해 있는 독자들은 보다 쉽고 빠르게 dBASE Ⅳ를 이해할 수 있으리라고 짐작하고, 여기에서는 dBASE Ⅳ의 사용을 망설였던 독자들의 판단을 돕기 위해 dBASE Ⅲ+와의 비교를 통해 dBASE Ⅳ에 새로이 개선된 기능들을 알아본다.

dBASE Ⅳ는 dBASE Ⅲ+에 기초를 두고 제작되었으며 특히 새로운 사용자나 멀티 유저를 위한 기능들이 추가되거나 보강되었고, dBASE Ⅲ+를 사용하면서 감수해야 했던 많은 불편한 문제들이 해결되는 등 dBASE 사용자들을 매료시키는 부분들이 많이 추가되었다.

새로운 사용자 인터페이스

dBASE Ⅳ는 dBASE를 처음 대하는 사용자도 거부감없이 사용할 수 있도록 편리하게 완전한 풀 다운 메뉴(Pull-down Menu) 방식을 제공한다. 그리고 강력한 윈도우 기능을 지원하면서 스크린 디자이너들을 위한 form, label 그리고 report의 방식으로 WY-SIWYG (What you see is what you get)방식을 채택하였다. 또한 form, label 그리고 report의 추적을 쉽게 하여 데이터베이스 화일의 유지보수를 쉽게 하기 위한 카탈로그의 기능도 대폭 향상되었다.

(1) 콘트롤 센터(Control Center)

이 기능은 dBASE Ⅲ+에는 없었던 새로운 기능으로 화일들을 열거나 닫고, 메뉴시스템을 운용하고, 프로그램을 실행시킬 수 있도록 고안되어 있으며, 화일과 카탈로그를 관리한다. 콘트롤센터는 현재의 카탈로그에 있는 모든 데이터베이스 화일들, views, forms, reports, labels 그리고 프로그램들을 한 화면에 모두 디스플레이시킨다. 또한 사용자의 요구에 따라 신속하게 그들 중의 어느 하나를 화면에 나타내기도 한다.

(2) Browse와 에디트

dBASE Ⅲ+에서는 분리되었던 이 두기능이 F2 데이터 키에 의해서 간단히 변환되며, 에디트 스크린에서 메뉴 바를 사용할 수 있고 browse에서의 메뉴 바의 기능이 크게 강화되었다. 또한 에디트 상태에서 커서를 옮기는 일도 보다 논리적이고 쉽게 할 수 있다.

(3) Form 기능

Screen painter의 기능이 대폭 강화되어 화면상에서 모든 필드와 필드 이름들을 자동적으로 배열시킬 수 있으며 각 요소들을 칼라로 표현할 수 있고 라인이나 박스 등도 쉽게 화면에 표현할 수 있다. 또 각각의 대상물들을 화면상의 원하는 위치에 쉽게 이동할 수 있는 등 화면편집기능의 서비스가 이용자들의 요구에 맞도록 보강되었다.

(4) 매크로 기능(Macro)

완벽한 키보드 매크로기능을 보유하고 있기 때문에 사용자의 요구가 있을 때면 언제든지 각종의 키스트록을 만들거나 기억시켜 둘 수 있다.

(5) Report 기능

Report는 dBASE Ⅲ+와는 다르게 새로 만들어졌으며, WYSIWYG 레이아웃에 의한 방식으로 밴드의 개념으로 레코드들을 처리할 수 있기 때문에 특정의 레코드들을 그룹으로 묶어서 처리한다. report를 만들때 사용자는 다섯가지의 summary fields(AVERAGE, SUM 등), 두가지의 에디팅 모드, 세가지의 자동 레이아웃 기능중에서 선택할 수 있다. 라인이나 박스 등을 자유롭게 사용할 수 있을 뿐만 아니라 report는 프린터나 디스크화일 어느 곳으로든 전송이 가능하다.

명령어와 함수

(1) CALCULATE

CALCULATE 명령을 사용하면 표준편차, 산술평균, 분산, 합계, 최소치, 최대치 등의 통계처리나 회계처리에 필요한 각종 수치들을 쉽게 취급할 수 있으며, 또한 CALCULATE는 CAL-CULATE 가 갖고 있는 어떤 함수도 데이터베이스를 통하여 단일패스로 처리할 수가 있어서 dBASE Ⅲ+보다 훨씬 빠른 속도로 계산할 수 있다.

예를 들어 보면 dBASE Ⅲ+의 경우 다음과 같이 할 때

```
USE customer
COUNT TO Nbr_Custs
SUM Total_Dol to
Amt_Spent
```

dBASE Ⅳ에서는 다음 리스트 1과 같이 한다.

앞의 예에서, COUNT와 SUM으로 분리되었던 명령이 CAL-CULATE 로 단일화되면서 속도도 훨씬 빨라진다는 것을 알 수 있다. 참고로 CALCULATE에서 사용할 수 있는 함수들은 다음과 같다.

AVR(expN) : expN의 산술평균치
CNT() : 레코드수를 카운트
MAX(exp) : 특정 데이터베이스 필드 중에서 가장 큰 수
MIN(exp) : 특정 데이터베이스 필드 중에서 가장 작은 수
STD(expN) : expN의 표준편차
SUM(expN) : 합계치

추가된 기능들

〔리스트 1〕

```
USE customer
CALCULATE CNT(), SUM(Total_dol) TO Nbr_Custs,Amt_Spent
```

〔리스트 2〕

```
SCAN FOR Due_Date = DATE() .AND. Amt_Due > 0
  DO Letter WITH LastName, FirstName, Amt_Due
ENDSCAN
```

〔리스트 3〕

```
LOCATE FOR Due_Date = DATE() .AND. Amt_Due > 0
DO WHILE FOUND() .AND. .NOT. EOF()
  DO Letter WITH LastName, FirstName, Amt_Due
  CONTINUE
ENDDO
```

VAR(expN) : expN의 분산

(2) 캐릭터

표현식과 매크로 확장에서 dBASE Ⅳ는 화일이름을 쓸 때 캐릭터 표현식이나 매크로 확장 모두를 사용할 수 있는데 가능하면 캐릭터 표현식을 사용하는 것이 매크로 확장을 사용하는 것보다 훨씬 속도가 빠르다.(경우에 따라서는 후자가 사용에 편리할 때도 있지만)예를 들면, dBASE Ⅲ+의 경우

```
STORE "Payroll1" to Mfilename
USE &Mfilename
```

와 같이 하는데 dBASE Ⅳ는 다음과 같이 처리하므로 속도가 빠르다.

```
STORE "Payroll" to Mfilename
USE Mfilename + "1"
```

(3) 수학적인 함수

산술계산과 삼각함수의 계산을 위하여 다음과 같은 함수들이 dBASE Ⅳ에 새로 추가되었다.

SIN() : 사인

ASIN() : 아크 사인

COS() : 코사인

ACOS() : 아크 코사인

TAN() : 탄젠트

ATAN() : 아크 탄젠트

ATN2() : 아크 탄젠트값을 각도로

SIGN() : 기호(+, −)

PI() : 3.14159

RTOD() : 라디안에서 디그리로 변환

DTOR() : 디그리에서 라디안으로 변환

CEILING() : 수식계산의 결과 가장 적은 값을 갖는 절대값

FLOOR() : 수식계산의 결과 가장 큰 값을 갖는 절대값

LOG10() : 상용대수

RAND() : 난수표의 발생

(4) SCAN / ENDSCAN 명령어

DO WHILE / ENDDO 루프와 같은 기능을 하면서도 보다 쉽고 적은 명령어로 루프기능을 수행할 수 있고, LOOP와 EXIT 명령어로 쉽게 SCAN 루프에 출입이 가능하다. 리스트 2, 3의 두가지 예는 동일한 기능을 수행한다.

메모리 변수와 Array

(1) Array

dBASE Ⅳ는 array의 사용이 가능하며, APPEND FROM ARRAY, AVERAGE, COPY TO ARRAY, DECLARE, PUBLIC 그리고 SUM 등의 모든 기능이나 명령의 사용이 가능하도록 되어 있다. dBASE Ⅳ에서의 array는 2차원까지 가능하고, array의 사용가능한 최대의 갯수는 컴퓨터의 메모리 용량에 따라 결정되며 최대 1,170개까지 사용이 가능하다.

(2) 메모리 변수

dBASE Ⅳ에서 사용가능한 변수의 수는 최대 15,000개이며, 변수명을 내부에서 찾는데 소요되는 시간을 많이 줄였다. 또한 수치 데이터 중에 부동소숫점(floating point)을 사용할 수 있는 F 타입이 새로 마련되었는데, 수식계산을 실행할 때 고정소숫점과 부동소숫점 모두를 사용하면 통상 부동소숫점 숫자로 처리된다. 그러나 FIXED() 함수에 의해서 부동소숫점 숫자가 고정소숫점 숫자로 바뀐 경우에는 예외이다.

보다 강력해진 기능들

(1) 인덱싱

dBASE Ⅳ는 .mdx(multiple index)라고 불리우는 오직 한개의 화일로 무려 47개의 인덱스 태그를 갖는데 각각의 인덱스 태그는 dBASE Ⅲ+에서의 .ndx와 유사하다. 물론 47개의 인덱스 태그 중 일부를 사용하거나 전부를 사용할 수 있으며, 또한 .mdx화일은 한개의 데이터베이스 화일에 여러개를 가질 수도 있는데 한번 open된 .mdx 화일은 데이터의 변동이 있을 때마다 자동적으로 업데이트되며 내림차순이나 올림차순 어느 방향으로도 인덱스가 가능하다.

가령 한개의 데이터베이스 화일이 세개의 인덱스를 갖는 경우를 예로 들어 알아보자. dBASE Ⅲ+에서 다음 리스트 4와 같이 한다면

dBASE Ⅳ의 경우는 다음 리스트 5와 같이 한다.

이 예에서 dBASE Ⅲ+는 인덱스화일을 업데이트시키기 위해서 USE Customer INDEX Custnbr, Custname, Custzip로 반드시 인덱스화일 세개 모두를 open시켜야 하지만, dBASE Ⅳ는 USE Customer라고만 하면 된다. dBASE Ⅳ에서 .ndx와

〔리스트 4〕

```
USE Customer
INDEX ON Cust_nbr TO Custnbr
INDEX ON SUBSTR(Last_Name,1,8)+SUBSTR(first_name,1,1) TO Custname
INDEX ON Zip_code+SUBSTR(Last_Name,1,8) TO Custzip
```

〔리스트 5〕

```
USE Customer
INDEX ON Cust_nbr TAG Custnbr
INDEX ON SUBSTR(Last_Name,1,8)+SUBSTR(first_name,1,1) TAG Custname
INDEX ON Zip_code+SUBSTR(Last_Name,1,8) TAG Custzip
```

.mdx 모두 사용이 가능하지만, 이 두가지 타입의 인덱스화일은 10개 까지만 사용할 수 있다.

(2) MODIFY COMMAND

dBASE Ⅲ+에서 프로그램화일(.prg)을 만들때 라인당의 길이와 편집할 수 있는 라인의 수가 적어 불편을 겪어야 했는데 dBASE Ⅳ는 라인당 1,024 캐릭터를 쓸 수 있고 최대 32,000 라인을 한개의 .prg 화일에 쓸 수 있다. 또한 에디터의 기능이 대폭 강화되어 블록을 만들어 복사를 하거나 이동할 수 있고, dBASE Ⅲ+의 제어코드를 그대로 사용할 수 있으며 풀다운 메뉴 방식으로 사용이 매우 편리해졌다.

(3) 다른 화일들과의 연결

dBASE Ⅱ, Framework Ⅱ, Rapidfile, MultiPlan, LOTUS 1-2-3 및 PFS 등을 읽거나 쓸 수 있다.

편리한 프로그래밍 기법

(1) 어플리케이션 제너레이터

dBASE Ⅲ+보다 기능이 훨씬 강력해진 어플리케이션 제너레이터는 dBASE의 새로운 사용자나 경험자에게 한줄의 프로그램 코드도 직접 작성하지 않아도 될 수 있을 만큼 강력한 메뉴 진행방식을 제공하므로 사용하기 쉽다. 어플리케이션 제너레이터는 콘트롤센터에서나 도트 프롬프트 상태에서 모두 사용할 수 있다.

(2) 컴파일러

별도의 컴파일러 패키지를 사용하여야 했던 dBASE Ⅲ+와는 달리 dBASE Ⅳ는 내장된 프로그램 컴파일러가 제공된다.

(3) 디버거

윈도우상태와 MODIFY COMMAND 상태에서 동작하는 디버거가 제공되는데 DEBUG <filename> 또는 SET TRAP ON으로 실행시킬 수 있으며, 디버그 윈도우는 프로그램화일, 프로시듀어 화일, 현재의 라인 번호, 작업영역, 데이터베이스, 인덱스 화일 그리고 현재의 레코드수 등 작업환경에 관한 정보를 화면에 표시한다. 또한 명령어가 수행되는 것을 추적할 수 있어서 작성된 프로그램의 디버깅이 용이하다.

(4) 메뉴

프로그래밍 언어를 사용하여 사용자가 메뉴를 직접 제작할 수 있으며, 메뉴명령어와 함수들은 사용자가 팝업 메뉴와 수평바를 쉽게 만들거나 수정할 수 있다. (팝업메뉴란 스크린에 윈도우를 만들어 주며, 수평 바를 통하여 선택할 수 있고, 선택되면 다음 메뉴로 진행을 하거나 명령어를 처리한다.)

dBASE Ⅳ에 새로 만들어진 메뉴에 관한 명령어와 함수들은 다음과 같다.

```
DEFINE POPUP
DEFINE BAR
ACTIVATE POPUP
SHOW POPUP
DEACTIVATE POPUP
CLEAR POPUPS
RELEASE POPUPS
ON SELECTION POPUP
POPUP()
BAR()
PROMPT()
DEFINE MENU
DEFINE PAD
ACTIVATE MENU
SHOW MENU
DEACTIVATE MENU
RELEASE MENU
CLEAR MENUS
ON PAD
ON SELECTION
MENU()
PAD()
```

(5) 프로시듀어

dBASE Ⅳ에서는 dBASE Ⅲ+와는 다르게 프로시듀어들을 처리하는데 이것은 작성된 소스 프로그램을 오브젝트 코드(.dbo file)로 컴파일할 때 프로시듀어를 만나면 명령어화일에 있는 오브젝트 화일의 프로시듀어 리스트에 추가되어야 하기 때문이다. 그러나 dBASE Ⅲ+에서 사용되는 SET PROCEDURE TO 명령은 사용이 가능하다. 또한 프로그램이나 프로시듀어가 사용할 수 있는 프로시듀어의 총 수량은 32개에서 963개로 크게 늘어났다.

(6) 사용자정의 기능

만일 dBASE Ⅳ에서 제공되지 않는 기능이 있다면 사용자가 정의하여 사용할 수 있으며 사용자가 정의한 기능도 dBASE Ⅳ가 제공하는 기능처럼 사용할 수 있다.

(7) 윈도우 기능

도트 프롬프트상태나 프로그래밍 언어에 의해 윈도우를 제작할 수 있는데 한번에 최대 20 개까지의 윈도우를 사용할 수 있으며 색상, 위치를 마음대로 선택할 수도 있고, 윈도우위에 다른 윈도우를 놓을 수도 있다.

데이터베이스 화일

(1) 향상된 제한조건

dBASE Ⅳ와 dBASE Ⅲ+의 제한 사항들을 비교하면 표 1과 같다.

(2) 프로텍트

PROTECT 유틸리티는 별도의 외부 유틸리티를 사용하지 않고 도트 프롬프트 상태나 콘트롤센터 상태에서 직접 사용이 가능하다.

(3) Read-Only 화일

단일 또는 멀티유저 모드에서 read-only 화일의 사용이 가능하기 때문에 데이터베이스를 보다 효율적으로 관리할 수 있다.

변경된 명령어와 기능들

dBASE Ⅳ에서 사용되는 명령어는 dBASE Ⅲ+에서 사용되던 명령어의 일부를 수정보완하였거나, 특정한 기능이나 새로운 명령어들이 추가되었다. 다음에서 설명되는 명령어나 function에서 사용된 기호들의 의미를 알아보자.

대문자 : dBASE Ⅳ 예약어

[] : 사용자 선택항목(필요에 따

	dBASE Ⅲ+	dBASE Ⅳ
데이터 화일		
레코드의 수	10억개	10억개
바이트 수	20억개	20억개
레코드 크기	4,000바이트	4,000바이트
필드 수	128개	255개
인덱스 화일		
인덱스 수	1개(.ndx당)	47개(.mdx당)
필드 사이즈		
캐릭터 필드	254바이트	254바이트
날짜 필드	8바이트	8바이트
논리 필드	1바이트	1바이트
수치 필드	15.9디지트	20디지트
부동소숫점 번호 필드	없음	20디지트
Arrays	없음	
디멘전		2
총 크기		1,170
화일 운용		
Open 화일(모든 종류)	15개	99개
Open DB 화일	10개	10개
Open 인덱스 화일	7개	10개
Open 포맷 화일	1개	1개
수치 정확도		
타입 F번호	없음	15.9자리 0.9*E+308 0.1*E−307
타입 N번호	15.9자리 1*E+99 1*E−307	20자리 0.9*E+308 0.1*E−307
메모리 변수	최대 256개	최대 15,000개

라 없어도 무방함)

/ : 둘 또는 그 이상에서 하나를 선택하여야 함.

〈〉: 사용자가 적당한 값을 입력하여야 함

〈expC〉: 캐릭터타입 데이터

〈expN〉: 수치(BCD)타입 데이터

〈expF〉: 부동소숫점타입 데이터

〈expD〉: 날짜타입 데이터

〈condition〉: 논리타입 데이터

(1) 화일에서의 출력기능

dBASE Ⅲ+에서는 화면이나 프린터로만 가능했던 출력을 화일에도 할 수 있도록 기능이 첨가된 명령어들은 다음과 같다.

- DISPLAY FILES [[LIKE] 〈skeleton〉] [TO PRINTER / TO FILE 〈filename〉]
- DISPLAY HISTORY [LAST 〈expN〉] [TO PRINTER / TO FILE 〈filename〉]
- DISPLAY MEMORY [TO PRINTER / TO FILE 〈filename〉]
- DISPLAY STATUS [TO PRINTER / TO FILE 〈filename〉]
- DISPLAY STRUCTURE [IN 〈alias〉] [TO PRINTER / TO FILE 〈filename〉]
- DISPLAY [[FIELDS] 〈expression list〉] [OFF] [TO PRINTER / TO FILE 〈filename〉] [〈scope〉] [WHILE 〈condition〉] [FOR 〈dondition〉]
- EJECT PAGE

(2) 새로운 명령어

- ??? 〈expC〉: 프린터 드라이브를 인스톨시키지 않고도 프린터에 제어코드를 전송할 수 있다.
- @ 〈rowl〉, 〈col1〉 FILL TO 〈row2〉, 〈col2〉 COLOR 〈attribute〉 : 사용자가 지정한 사각형의 범위를 사용자가 색으로 칠해준다.
- @ 〈row1〉, 〈col1〉 TO 〈row2〉, 〈col2〉 [DOUBLE / PANEL / 〈border definition string〉] [COLOR 〈color attribute〉]: 사용자가 지정한 범위에 사각형을 그리고 또한 색을 지정할 수 있다.
- ACTIVATE MENU 〈menu name〉 [PAD 〈pad name〉]: 이미 정의된 메뉴를 동작시킨다.
- ACTIVATE POPUP 〈popup name〉 : 이미 정의된 팝업 메뉴를 동작시킨다.
- ACTIVATE SCREEN : 이미 동작중인 스크린을 화면에 표시한다.
- ACTIVATE WINDOW 〈window name list〉 / ALL : 지정한 윈도우의 일부 또는 전부를 화면에 직접 출력한다. 윈도우가 동작중일 때 모든 입·출력은 윈도우를 이용하게 된다.
- COPY TO 〈filename〉 [FOR 〈condition〉] [[TYPE] DBASEII / DBMEMO3 / SDF / DIF / WKS / SYLK/RPD/FW2/DELIMITED [WITH BLANK/〈DELIMITER〉]] :Framework II, LOTUS 1-2-3, RapidFile, dBASE II 및 dBASE Ⅲ+ 메모 등을 액세스한다.
- CREATE APPLICATION 〈filename〉 / ? : 도트 프롬프트 상태에서 어플리케이션 제너레이터를 동작시킨다.
- DBLINK 〈filename〉 / L : 컴파일한 후 생긴 오브젝트 화일들을 링크시킨다.
- DEACTIVATE MENU : 동작중인 메뉴의 동작을 중지시키고 화면에서 지운다. 그러나 메모리의 내용은 지워지지 않는다.
- DEACTIVATE POPUP : 동작중인 팝업의 동작을 중지시키고 화면에서 지운다. 그러나 메모리의 내용은 지워지지 않는다.
- DEACTIVATE WINDOW 〈window name list〉 / ALL : 윈도우의 일부 또는 전부의 동작을 중지시킨다.
- DEBUG 〈filename〉 / 〈procedure name〉 [WITH 〈parameter list〉] : 디버거를 실행시킨다.
- DECLARE 〈array element list〉 : array를 만든다.
- DEFINE BAR 〈line number〉 OF 〈popup name〉 PROMPT 〈expC〉 [MESSAGE 〈expC〉 [SKIP[FOR 〈condition〉]] : 팝업 메뉴에서 바를 선언한다.
- DEFINE BOX FROM 〈print column〉 TO 〈print column〉 HEIGHT 〈expC〉 [AT LINE 〈print line〉] [SINGLE / DOUBLE / 〈border definition string〉] : 프린터에 그래픽으로 사각형을 만든다.
- DEFINE MENU 〈menu name〉 [MESSAGE 〈expC〉] : 메뉴를 선언한다.
- DEFINE PAD 〈pad name〉 OF 〈menu name〉 PROMPT 〈expC〉 [AT 〈row〉, 〈col〉] [MESSAGE 〈expC〉] : pad를 선언한다.
- DEFINE POPUP 〈popup name〉 FROM 〈row1〉, 〈col1〉 [TO 〈row2〉, 〈col2〉] [PROMPT FIELD 〈field name〉 / PROMPT FILES [LIKE 〈skeleton〉] / PROMPT STRUCTURE] [MESSAGE 〈ex-

pC〉] : 팝업 메뉴를 선언한다.
- DEFINE WINDOW 〈window name〉 FROM 〈row1〉, 〈col1〉 TO 〈row2〉, 〈col2〉 [DOUBLE / PANEL / NONE / 〈border definition string〉] [COLOR [〈standard〉] [,〈enhanced〉] [,〈frame〉]] : 윈도우를 선언
- DELETE TAG 〈tag name1〉 [OF 〈.mdx filename〉] / 〈.ndx filenamel〉 [〈.tag name2〉 [OF 〈.mdx filename〉] / 〈.ndx filename2〉...] : 지정한 인덱스 태그를 지정한 .mdx화일로부터 지우거나, 지정한 .ndx 화일들을 닫는다.
- FUNCTION 〈procedure name〉 : 사용자 정의 함수
- LOGOUT : 모든 화일을 닫고, 내부의 dBASE Ⅳ 파라미터들을 리세트시킨다.
- MOIFY APPLICATION 〈filename〉 : 어플리케이션 화일을 수정
- ON PAD〈pad name〉 OF〈menu name〉 [ACTIVATE POPUP〈popup name〉] : pad를 지정한다.
- ON PAGE [AT LINE 〈expN〉 〈command〉] : 프린터에서 페이지 넘길 곳을 지정
- ON READERROR [〈command〉] : 풀 스크린 동작중 발생되는 에러를 찾아서 수정하기 위함.
- ON SELECTION PAD 〈pad name〉 OF 〈menu name〉[〈command〉] : pad가 눌러졌는가를 확인하고 눌러지면 그에 해당하는 명령을 실행.
- ON SELECTION POPUP 〈popup name〉 / ALL [〈command〉] : 팝업 메뉴가 선택되었는가 확인하고 해당 명령을 실행.
- PLAY MACRO 〈macro name〉 : 매크로를 실행.
- PRINTJOB / ENDPRINTJOB : 인쇄에 관한 구조적 프로그래밍 명령
- PROTECT : 내부명령으로 프로텍트를 정의 또는 수정
- RELEASE MENUS [〈menu name list〉] : 메뉴들을 지움.
- RELEASE POPUP [〈popup name list〉] : 팝업들을 지움
- RELEASE WINDOWS [〈window name list〉] : 윈도우를 지움
- RESET [IN 〈alias〉] : 태그들을 지움.
- RESTORE MACROS FROM 〈filename〉 : 특정 화일로부터 매크로를 회복
- RESTORE WINDOW 〈window name list〉 / ALL FROM 〈filename〉 : 윈도우를 회복
- ROLLBACK [〈database filename〉] : 현재 운용중인 데이터베이스(인덱스 포함)보다 먼저 사용되었던 데이터베이스를 복구시킴
- SAVE MACROS TO 〈macro file〉 : 정의된 매크로들을 지정한 화일에 저장
- SAVE WINDOW 〈window name list〉 / ALL TO 〈filename〉 : 윈도우를 저장
- SCAN [〈scope〉] [FOR 〈condition〉] [WHILE 〈condition〉] [commands〉] [EXIT] [LOOP] ENDSCAN : 루프를 구성
- SHOW MENU 〈menu name〉 [PAD 〈pad name〉] : 이미 정의된 메뉴를 화면에 출력
- SHOW POPUP 〈popup name〉 : 팝업을 화면에 표시
- UNLOCK [ALL / IN 〈alias 〉] : 록되어 있는 레코드나 화일을 풀어줌

(3) 함수기능

dBASE Ⅳ에는 여러가지 함수가 첨가되었는데 다음에서 간단하게 알아보자.

- ACCESS() : 사용자의 액세스 레벨(1 ~ 8)을 얻는다.
- ACOS(〈expN〉) : 아크코사인(라디안)
- ALIAS([〈expN〉]) : alias 이름을 얻는다.
- ASIN(〈expN〉) : 아크사인 (라디안)
- AT(〈expC〉, 〈expC〉 / 〈memo field name〉) : 첫번째 스트링이 두번째 스트링을 포함하는가, 하지 않는가를 확인
- ATAN (〈expN〉) : 아크탄젠트(라디안)
- ATN2(〈expN1〉, 〈expN2〉) : 아크탄젠트(사인 / 코사인) 값을 각도로 나타낸다.
- BAR() : 최근에 선택했던 바의 번호를 얻는다.
- CALL(〈expC〉, 〈expC〉 / 〈memvar name〉) : 바이너리 프로그램 모듈에 사용
- CEILING(〈expN〉) : 수식계산의 결과 가장 적은 값을 갖는 절대값
- CHANGE() : 현재의 레코드가 수정되었는가 확인
- COMPLETED() : 전송이 끝났는 가를 확인
- COS(〈expN〉) : 코사인(라디안)
- DIFFERENCE(〈expC〉, 〈expC〉) : 두개의 리터럴 스트링이 얼마나 다른가를 숫자로 나타낸다.
- DMY(〈expD〉) : 년월일을 캐릭터 스트링으로 나타낸다.
- DTOR(〈expN〉) : 삼각함수의 디그리를 라디안으로 변환시킨다.
- FIXED (〈expN〉) : 수치 데이터 타입 F(부동소숫점)을 타입 N(고정소숫점)으로 변환시킨다.
- FLOAT(〈expN〉) : 수치 데이터 타입 N(고정소숫점)을 타입 F(부동소숫점)로 변환시킨다.
- FLOOR(〈expN〉) : 수치계산의 결과 가장 큰 값을 갖는 절대값
- FV(〈payment〉, 〈rate〉, 〈periods〉) : 저축에 대한 이자 금액의 계산
- INKEY([〈n〉]) : 최근에 눌렀던 키를 숫자로 얻는다.
- ISMARKED([〈alias〉]) : 데이터베이스 화일 헤더가 변경되었는가를 확인
- KEY([〈.mdx filename〉,]〈expN〉] [,〈alias〉]) : 인덱스 화일의 키 연산자를 얻는다.
- LASTKEY() : 가장 최근에 눌렀던 키를 얻는다.
- LEFT(〈expC〉 / 〈memofield name〉, 〈expN〉) : 좌측으로부터 지정한 수만큼의 문자를 얻는다.
- LEN(〈expC〉 / 〈memofield name〉) : 문자열의 길이를 얻는다.
- LIKE(〈pattern〉, 〈expC〉) : 프로그램 함수로 와일드카드 비교에 사용함
- LINENO() : 디버거에서 사용되며, 화일이나 명령의 라인번호를 얻는다.
- LKSYS(〈n〉) : 사용자의 로그인에 관한 정보를 얻는다.
- LOG10(〈expN〉) : 상용로그값을 계산한다.
- MDX(〈expN〉 [,〈alias〉]) : 지정한 태그번호가 있는 .mdx 화일의 이름을 얻는다.
- MDY(〈expD〉) : 지정한 일자를 월일년의 순서로 만든다(예 : May 03, 1989).
- MEMORY([0]) : 남아있는 메모리의 크기를 나타낸다.
- MENU() : 현재 동작중이거나 최근에 동작중이었던 메뉴의 이름을 얻는다.

- NDX(〈expN〉 [,〈alias〉]) : 지정한 작업영역에서 동작중인 인덱스 화일의 이름을 얻는다.
- NETWORK() : 네트워크운용을 위해 dBASE Ⅳ가 설치되었는가를 확인
- ORDER([〈alias〉]) : 일차 인덱스화일이나 .mdx 태그의 이름을 얻는다.
- PAD() : 최근에 선택되었던 수평바의 이름을 얻는다.
- PAYMENT(〈principal〉, 〈rate〉, 〈periods〉) : 일정 기간에 대한 부채의 상환 금액을 계산
- PI() : 3.1415926535897932385
- POPUP() : 동작중인 팝업 메뉴의 이름을 얻는다.
- PRINTSTATUS() : 프린터상태가 준비상태인가 확인
- PROGRAM() : 에러가 발생했던 프로그램이나 프로시듀어의 이름을 얻는다.
- PV(〈payment〉,〈rate〉, 〈periods〉) : 수익금을 계산해준다.
- RAND([〈expN〉]) : 난수표를 발생한다.
- RIGHT(〈expC〉 / 〈variable〉, 〈expN〉) : 우측으로부터 지정한 수만큼의 문자를 얻는다.
- ROLLBACK() : ROLLBACK 명령이 실행되었는지를 확인
- RTOD(〈expN〉) : 삼각함수에서 라디안을 디그리로 변환시킨다.
- SEEK(〈expC〉 [,〈alias〉]) : FIND / SEEK와 FOUND()를 결합 한 것.
- SELECT() : 사용되지 않은 최상위의 작업 영역 번호를 얻는다.
- SET(〈expC〉) : SET TO와 SET ON 명령어의 상태를 확인한다.
- SIGN(〈expN〉) : 양수이면 1, 0이면 0, 음수이면 -1을 얻는다.
- SIN(〈expN〉) : 사인값을 얻는다.
- SOUNDEX(〈expC〉]) : 철가 점검을 위해 사용됨
- SUBSTR(〈expC〉 / 〈memo field name〉, 〈starting position〉 [, 〈number of characters 〉]: 특정의 문자열에서 특정의 문자를 얻는다.
- TAG([〈.mdx filename〉,] 〈expN〉] [,〈alias〉]) : .mdx 화일의 태그이름을 얻는다.
- TAN(〈expN〉) : 탄젠트값을 얻는다.
- USER() : 프로텍트된 시스템에서 현재 등록된 사용자 이름을 얻는다.

(4) 화일헤더

dBASE Ⅳ의 화일헤더구조는 표 2와 같다.

바이트	구 성	의 미
0	1바이트	비트 0~2 : 버전 번호 비트 3~5 : SQL의 예약 비트 6~7 : 메모화일 표시
1~3	3바이트	화일이 수정된 날짜
4~7	32비트 번호	레코드의 수
8~9	16비트 번호	헤더의 바이트 수
10~11	16비트 번호	한 레코드의 길이(바이트)
12~13	2바이트	예약
14	1바이트	ISMARKED( ) 함수 체크 플래그
15	1바이트	Encryption 플래그
16~27	12바이트	멀티 유저를 위함
28	1바이트	Production.mdx' 화일 플래그
29~31	3바이트	예약
32~n	각 32바이트	11바이트 : 필드 이름(ASCII) 1바이트 : 필드 타입(C, D, F, L, M 또는 N) 32비트 번호 : 필드 데이터 어드레스 1바이트 : 필드 길이(2진수) 1바이트 : 필드의 소숫점 자리수(2진수) 2바이트 : 예약 1바이트 : 작업영역 ID 11바이트 : 예약
n+1	1바이트	필드 종료(0D : 16진수)

한글의 사용

dBASE Ⅳ에서는 2바이트 조합형 한글(예 : nkp.com 등)은 사용할 수 없으나 대부분의 독자들이 사용하고 있는 청계천 세운상가의 한글카드는 한글사용에 문제가 없다.

또한 행정전산망에서 사용되고 있는 공업진흥청 표준한글 코드인 2바이트 완성형 한글도 사용하는데 문제가 없다. 다만, IBM PC-XT를 사용하여 한글을 소프트웨어적으로 처리할 때는 주기억장치의 용량이 문제가 된다. K-한글의 경우는 시동할때 K / L16 : 1024 / B16 : 128로 해야만 메모리 부족으로 인해서 dBASE Ⅳ가 동작하지 않는 것을 피할 수 있다.

맺는말

이제까지 설명했던 dBASE Ⅳ의 문제점은 반드시 하드디스크를 사용하여야 하는 것이다. 최근에 소개되고 있는 상당수의 패키지들 (WordPerfect 5.0, SPSS / PC+)이 하드디스크의 사용을 요구하고 있고 dBASE Ⅳ의 경우도 예외는 아니기 때문에 하드디스크를 준비하지 못한 독자에게는 유감이 아닐 수 없다. 그러나 하드디스크의 가격이 점차로 하락하는 추세이므로 이번 기회를 빌어 하드디스크를 준비하여 dBASE Ⅳ와 같은 유용한 소프트웨어를 사용할 수 있다면 그만한 투자가치를 얻을 수 있으리라는 확신을 가지고 독자 여러분께 감히 하드디스크의 사용을 권한다.

앞에서 dBASE Ⅲ+와 비교하면서 dBASE Ⅳ를 간략하게나마 살펴본 바와같이 dBASE Ⅳ는 dBASE Ⅲ+의 사용상의 문제점들을 많이 개선하면서 그 기능을 강화시켰음을 알 수 있다. 지면 관계상 자세한 설명없이 개괄적인 설명을 하다보니 초보자에게는 이해하는데 어려움을 주었을 것이다. 차후에 기회가 주어지면 dBASE Ⅳ의 진수를 맛볼수 있는 기회를 갖기로 하고, 우리나라에서도 외국의 것보다 훨씬 좋은 소프트웨어들이 많이 개발, 보급되기를 기대해 본다.

GET과 PUT의 100% 활용
그래픽 에니메이터

베이직언어는 컴퓨터 그래픽을 손쉽게 구현할 수 있는 좋은 명령어들을 가지고 있다. 물론 이 외에도 파스칼이나 C, 또는 어셈블리어를 통해서도 훌륭한 그래픽을 얻을 수 있지만 이들 언어를 아직 익히지 못한 독자라면 풍부한 그래픽 명령을 구비하고 있으면서 배우기도 쉬운 베이직을 권하고 싶다.

이미 베이직을 통해 그래픽을 구현하고 있는 독자들도 있겠지만 여기서는 특히 에니메이션에 대해서 관심을 가져보기로 하겠다. 그래픽 명령을 통한 에미메이션의 원리를 간단히 살펴보면 먼저 원하는 그래픽 이미지를 그린 다음 특정한 배열이름으로 그 이미지를 기억시킨다. 그런 다음 원하는 위치(좌표)에서 배열이름을 지정하여 이미지를 출력하는 것으로 이 위치를 적절하게 변화시키므로써 더욱 자연스러운 움직임을 얻을 수 있다.

베이직(여기서는 GWBASIC 이용) 매뉴얼을 보면 에니메이션, 즉 움직이는 이미지를 만들어내는 그래픽명령으로 GET과 PUT이 있다. GET명령은 그래픽 이미지를 저장하기 위한 것으로 프로그래머가 지정한 사각형 안의 픽셀값(더 정확하게 말하면 픽셀이 갖는 컬러값)을 지정한 배열에 저장하는 기능을 가진다. 여기서 사각형이란, 이동시키고자 하는 이미지가 있을 때 그 이미지를 포함하는 사각형태를 말하며 실제 명령에서는 좌측 상단 꼭지점인 (X1, Y1)과 우측 하단 꼭지점인 (X2, Y2), 그리고 이들 두 꼭지점을 대각선으로 하는 사각형 안의 픽셀값들을 저장하기 위한 배열로써 이루어진다.(그림 1 참조).

이와같이 GET명령을 이용하기 전에 미리 배열명을 선언해야 하는데 이 배열은 바이트 단위로 기억될 수 있는 수치형 배열이어야 한다. (통상 베이직에서는 수치형 변수나 배열명을 나타내기 위해 변수명 다음에 '%'를 붙인다.) 그리고 배열의 크기를 정할 때는 다음과 같은 방법으로 바이트 수를 구하는 것이 효율적이다.

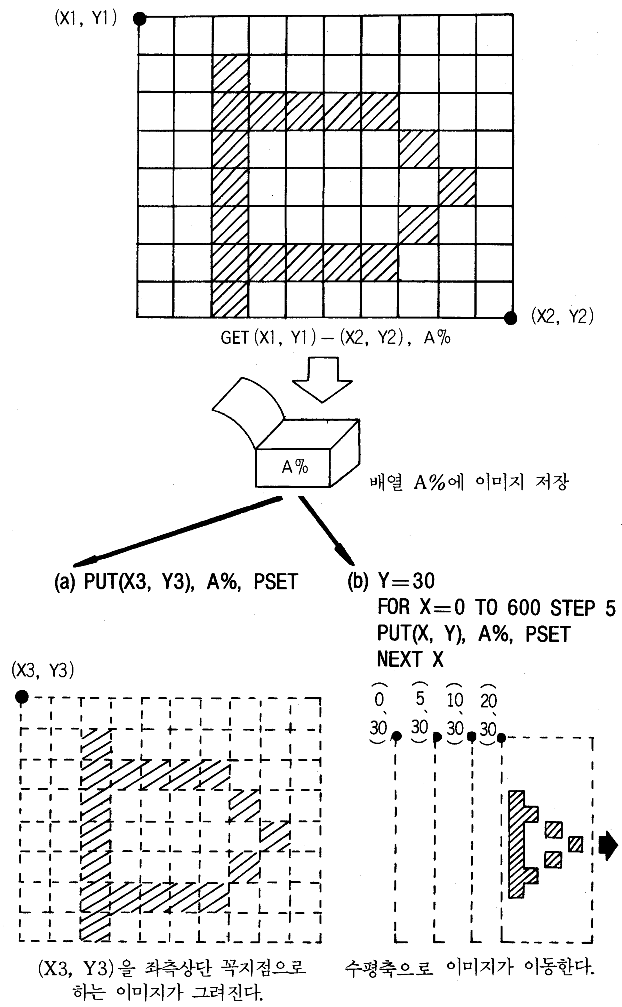
```
바이트 수  = 4 + INT[(X*B + 7)/8]* Y
```

여기서 X는 이미지를 포함하는 사각형의 수평축 비트수를 나타내고 Y는 세로축의 비트수를, 그리고 B는 픽셀당 비트수를 나타낸다. 고해상도에서 1비트가 1픽셀인데 비해 중해상도에서는 2비트가 모여 1픽셀을 이루기 때문에 B의 값은 고해상도에서 1, 중해상도에서는 2가 된다. 예를 들어 다음과 같은 GET명령을 실행할 때

```
GET (10,10) - (19,19) , 배열명
```

중해상도 모드에서는 34바이트 크기를 갖는 배열이 미리

〔그림 1〕 에니메이션 원리



선언되어 있어야 함을 알 수 있다.

```
4 + INT[(10*2 + 7)/8]*10 = 34(바이트)
```

이렇게 하여 사각형 형태로 이미지가 저장되었다면 PUT 명령으로 원하는 좌표에서 다시 그려 낼 수 있다. PUT명령은 그려질 위치의 좌측 상단 좌표와 출력 효과를 지정하는 파라미터로 구성된다. 이용가능한 파라미터는 PSET, PRESET, XOR, OR, AND 등이며 아무런 지정이 없으면 디폴트 값으로 XOR이 선택된다.

PSET을 선택하면 기존의 이미지를 지우며 그려지는 효과를 얻는 반면 PRESET은 이미지를 반전시켜 그리는데 이용될 수 있다. 일반적으로 중해상도 모드에서는 컬러코드 0과 3, 그리고 1과 2로 반전효과를 얻고 있으며 고해상도 모드에서는 0과 1을 이용하고 있다. 그 밖의 AND, XOR, OR은 배열에 기억된 컬러와 화면의 컬러를 서로 다른 방법으로 섞어가며 출력시킨다. 표 1을 보면 PSET은 기억된 그대로 그려주고 PRESET은 반전된 형태로, 그리고 AND나 OR 등은 나름대로의 논리에 의해 색을 섞어가며 그려주고 있음을 볼 수 있다.

에니메이터(ANIMATOR)

리스트 1의 ANIMATOR.BAS는 만화가가 쓰는 스케치패드와 같이 20개의 프레임에 연속적인 이미지를 그린 다음, 처음 순서부터 빠른 속도로 출력시켜 움직임을 표현하고 있으며 이미 완성한 이미지는 기억시켜 두었다가 다시 로드시킬 수도 있고 에니메이션되는 시작 프레임과 끝 프레임을 이용자가 마음대로 지정(예를 들어 1번부터 20번 프레임까지의 긴 연속물에서 1번에서 3번까지의 짧은 움직임만 관찰할 수도 있다.)할 수 있도록 하여 GET과 PUT 명령을 익히기에 적합한 프로그램이라고 본다.

배열 선언(180행)

20개의 각 프레임에 144바이트 크기를 갖는 정수형 배열을 A%, B%, C%, D%, E%...등으로 선언하였다.

```
180 DIM A%(144),B%(144),C%(144), .. T%(144),U%
    (144),A(20,54)
```

〔표 1〕 PUT 명령에서의 파라미터 효과

배열 컬러	PSET 화면컬러 0 1 2 3	PRESET 화면컬러 0 1 2 3	AND 화면컬러 0 1 2 3	OR 화면컬러 0 1 2 3	XOR 화면컬러 0 1 2 3
0	0 0 0 0	3 3 3 3	0 0 0 0	0 1 2 3	0 1 2 3
1	1 1 1 1	2 2 2 2	0 1 0 1	1 1 3 3	1 0 3 2
2	2 2 2 2	1 1 1 1	0 0 2 2	2 3 2 3	2 3 0 1
3	3 3 3 3	0 0 0 0	0 1 2 3	3 3 3 3	3 2 0 1

여기서 A(20, 54)는 20✱54 크기를 갖는 2차원 배열로서 디자인 격자의 이미지를 기억하기 위해 쓰인다.

키 입력대기(260행~400행)

ANIMATOR.BAS를 실행시킨 상태에서 'A$=INKEY$'를 통해 선택 키를 입력받고 올바른 키인지를 체크한다.

```
290 A$=INKEY$:IF A$="" THEN 290 ELSE IF LEN(A$)=2 THEN
    830 ELSE IF VAL(A$)>0 AND VAL(A$)<10 THEN 430
300 IF A$="P" THEN 3080
310 IF A$<>"Q" THEN 350
 :         :
380 IF A$="S" THEN 870 ELSE IF A$="L" THEN 990
390 IF A$="D" THEN 2800 ELSE IF A$="I" THEN 2360
```

이와같이 문자인 경우를 체크하고 430번 행에서는 숫자인 경우를 체크하여 해당 루틴으로분기시킨다.

```
430 ON VAL(A$)GOTO 520,1080,470,500,480,450,410,420,3040
```

즉, 1번은 에니메이션, 2번은 이미지 그리기, 3번은 시작 프레임 선택... 등을 지정하고 있는데 몇가지 중요한 루틴들을 살펴보면 다음과 같다.

이미지그리기(1080행)

초기화면 상태에서 '2'를 선택하고 <ENTER>키를 입력하면 잠시후에 커서가 깜박거리는 디자인 격자가 나타난다. 이 에디트 모드에서는 서로 다른 기능을 갖는 3개의 키를 이용할 수 있다. 'D'는 커서를 그릴 때 사용되고 'E'는 커서를 지울 때 사용되며 'M'은 커서를 그리지 않고 이동만 시킬 때 사용된다. 그리고 디자인 격자 왼쪽을 보면 실제 크기의 모양이 그려지고 있을 것이다. 이미지가 완성되면 'S'를 눌러 기억시킨 후 메인 메뉴로 돌아간다. 만일 에디터 모드에서 빠져나가고 싶다면 'Q'를 입력하면 된다.

```
1080 INPUT "EDIT PICTURE NUMBER";B:IF B<0 OR B>21 THEN
     BEEP:GOTO 1080
1090 IF B=21 THEN CLS:GOSUB 2310:GOTO 190
1100 IF B=0 THEN B=NUM
1110 '------- PUT PICTURE TO EDIT ON SCREEN -----------
1120 CLS:ON NUM GOTO 1130,1140,1150,1160,1170,1180,1190,
     1200,1210,1220,1230,1240,1250,1260,1270,1280,1290,
     1300,1310,1320
1130 PUT(1,50),A%:GOTO 1340
1140 PUT(1,50),B%:GOTO 1340
1150 PUT(1,50),C%:GOTO 1340
 :           :
1320 PUT(1,50),T%:GOTO 1340
```

위에서 1120행은 선택된 프레임 번호의 이미지를 화면 좌

측, 정확히 말하면 (1, 50)을 좌측 상단 좌표로 하는 사각형 안에 그리고 화면 중앙에서 디자인되는 격자 안의 내용을 2차원 배열 A(20, 54)에 저장시키는 과정은 다음과 같다.

```
1340 FOR I=1 TO 20
        FOR X= 1 TO 54
        A(I,X)=POINT(X,I+49)
        NEXT X
1350 NEXT I
```

디자인 격자에서 이미지를 그리는 메인 루프(MAIN LOOP)는 1570행부터 시작되는데 먼저 ROW와 COLUMN 변수에 '1'을 세트하고 CURS와 BLINK%에 '0'을 세트한 후 CURS의 값을 가지고 각 좌표를 ON, 또는 OFF시키고 있다. 즉 CURS의 값이 −1일 때는 '(E)RASE'상태로 해당 좌표값을 0으로 세트하고 화면의 픽셀을 지우며

```
A(ROW, COLUMN)=0
PSET(COLUMN, ROW+49),0
```

CURS의 값이 +1인 경우는 '(D)RAW'상태로 해당 좌표값을 1로 세트하고 화면의 픽셀을 그린다.

```
A(ROW, COLUMN)=1
PSET(COLUMN, ROW+49),1
```

이와는 달리 CURS의 값이 0인 경우는 '(M)OVE'상태로 픽셀의 ON, OFF와는 무관하게 커서의 이동만을 수행한다. 다음, 1420행의 서브루틴을 실행한 후 다시 입력키(A$)를 기다려 CURS의 값을 결정한다(그림 2 참조).

```
1620 IF CODE1=ASC("E") THEN CURS=-1
1630 IF CODE1=ASC("M") THEN CURS= 0
1640 IF CODE1=ASC("D") THEN CURS=+1
  :              :
1680 IF CODE1=ASC("I") THEN 1710
1710 GET(1,50)-(54,70),U%:PUT (1,50),U%,
     PRESET:GET(1,75)-(54, 95),U%
```

이미지 기억하기(2090행)

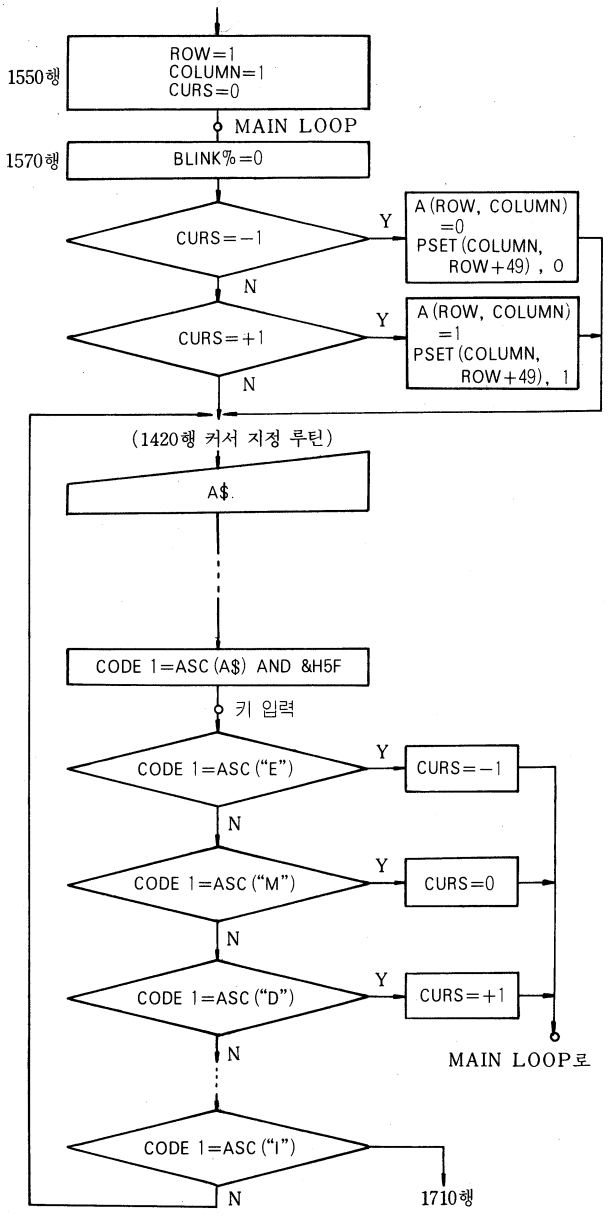
디자인 격자를 통해 한 프레임의 이미지가 완성되면 'S'옵션을 눌러 해당 번호의 프레임에 기억시킨다. 즉 2100행에서 해당 프레임의 번호를 찾아 2110행부터 나열된 해당 배열명에 GET명령으로 이미지를 기억한다(화면 1은 5개의 이미지를 만든 예를 보인 것이다.)

```
2100 ON B GOTO 2110,2120,2130,2140,2150, ...
2110 GET(1,50)-(54,70),A%: GOSUB 2310
2120 GET(1,50)-(54,70),B%: GOSUB 2310
2130 GET(1,50)-(54,70),C%: GOSUB 2310
  :                :
```

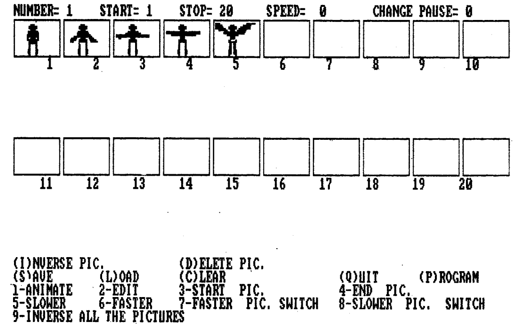
애니메이션 실행하기(520행)

애니메이션은 이미지를 갖는 첫번째 프레임번호와 마지막 프레임 번호를 각각 STA와 EN변수에 할당하여 STA부터 EN까지 QQ만큼 우측으로 이동하며 실행된다. 여기서 이미지가 교체되는 시간 간격은 FAST변수를 이용하여 570행에서 지연루프를 만들어 이용한다.

〔그림 2〕 키의 입력과 CURS의 처리 순서도



```
520 IF STA>EN THEN QQ=-1 ELSE QQ=1
530 FOR I=STA TO EN STEP QQ
 :              :
560 ON I GOTO 630,640,650,660,670, ...
570 FOR N=0 TO FAST*4:NEXT
 :              :
630 PUT(PL1,52),U%,PSET:PUT(PL,52),A%:GOTO 570
640 PUT(PL1,52),U%,PSET:PUT(PL,52),B%:GOTO 570
650 PUT(PL1,52),U%,PSET:PUT(PL,52),C%:GOTO 570
 :              :
 :              :
```



〔화면 1〕 다섯개의 이미지를 기억시킨 예

여기에서 알 수 있는 바와 같이 에니메이션을 선택하기 전에 시작프레임과 끝 프레임 번호를 꼭 지정해야 하며 이들 수치의 세팅이 완료되면 1번을 입력하여 연속물을 관찰한다.

〈SPACE BAR〉를 누르면 중지되고 완전히 종료시킬때는 〈ENTER〉키를 누른다.

〔리스트 1〕

```
10 'Program ANIMATOR.BAS ------------------------------
150 '------- MAIN PROGRAM -----------------------------
160 CLS:SCREEN 2:KEY OFF:FOR I=1 TO 10:KEY I,"":NEXT
170 '------ SET UP VARIABLES --------------------------
180 DIM A%(144),B%(144),D%(144),E%(144),F%(144),C%(
    144),G%(144),H%(144),I%(144),J%(144),K%(144),L%
    (144), M%(144), N%(144),O%(144),P%(144),Q%(144)
185 DIM R%(144), S%(144), T%(144),U%(144),A(20,54):
      NUM=1:STA=1:EN=20:SP=0:PL=1:GET(1,10)-(54,30),U%
190 X=9:Y=31:LOCATE 1,39:PRINT"SPEED= ";SP;:LOCATE 1,1
    :PRINT"NUMBER=";NUM;:LOCATE 1,14:PRINT"START=";STA;
    :LOCATE 1,26:PRINT"STOP=";EN;:LOCATE 1,55:PRINT
    "CHANGE PAUSE=";FAST
200 '------- SET UP SCREEN ------------------------
210 LOCATE 22,1:PRINT"1-ANIMATE   2-EDIT   3-START
    PIC.   4-END  PIC.":LOCATE 23,1:PRINT"5-SLOWER
    6-FASTER  7-FASTER PIC.SWITCH   8-SLOWER  PIC.
    SWITCH";:LOCATE 24,1:PRINT"9-INVERSE ALL ";
220 LOCATE 21,1:PRINT"(S)AVE    (L)OAD     (C)LEAR
    (Q)UIT    (P)ROGRAM";:LOCATE 4,78:FOR I=1 TO 10:
    PRINT"     ";I;:NEXT:LOCATE 13,79:FOR I=11
    TO 20:PRINT "    ";I;:NEXT
230 LOCATE 20,1:PRINT"(I)NVERSE PIC.          (D)EL
    ETE PIC.";
240 FOR T=1 TO 2:FOR I=1 TO 5:LINE(A,X)-(A+55,Y),,
    B:LINE(A+60,X)-(A+115,Y),,B:A=A+120:NEXT:A=0:X
    =80:Y=102:NEXT
250 '------- READ PICTURES -----------------------
260 GET(1,10)-(54,30),A%:GET(61,10)-(114,30),B%:GE
    T(121,10)-(174,30),C%:GET(181,10)-(234,30),D%:
    GET(241,10)-(294,30),E%
265 GET(301,10)-(354,30),F%:GET(361,10)-(414,30),G
    %:GET(421,10)-(474,30),H%:GET(481,10)-(534,30)
    ,I%:GET(541,10)-(594,30),J%
270 GET(1,81)-(54,101),K%:GET(61,81)-(114,101),L%:
    GET(121,81)-(174,101),M%:GET(181,81)-(234,101)
    ,N%:GET(241,81)-(294,101),O%
275 GET(301,81)-(354,101),P%:GET(361,81)-(414,101)
    ,Q%:GET(421,81)-(474,101),R%:GET(481,81)-(534,
    101),S%:GET(541,81)-(594,101),T%
280 '------ WAIT FOR KEY ------------------------
290 A$=INKEY$:IF A$="" THEN 290 ELSE IF LEN(A$)=2
    THEN 830 ELSE IF VAL(A$)>0 AND VAL(A$)<10 THEN
    430
300 IF A$="P" THEN 3080
310 IF A$=<>"Q" THEN 350
320 LOCATE 18,1:PRINT"ARE YOU SURE YOU WANT TO QUI
    T?(Y/N)"
330 A$=INKEY$:IF A$="" THEN 330 ELSE IF A$="Y" THE
    N SCREEN 0,0,0:CLS:END ELSE LOCATE 18,1: PRINT
    STRING$(50,32):GOTO 290
340 '------ CLEAR SCREEN ------------------------
350 IF A$<>"C" THEN 380
360 LOCATE 17,1:PRINT"ARE YOU SURE (Y/N)":DEF SEG:
    POKE 106,0
370 A$=INKEY$:IF A$="" THEN 370 ELSE IF A$="Y" THE
    N CLS:GOTO 190 ELSE LOCATE 17,1:PRINT STRING$(
    66,32):GOTO 290
380 IF A$="S" THEN 870 ELSE IF A$="L" THEN 990
390 IF A$="D" THEN 2800 ELSE IF A$="I" THEN 2360
400 GOTO 290
410 IF FAST=0 THEN BEEP:GOTO 290 ELSE FAST=FAST-1:
```

```
    LOCATE 1,68:PRINT FAST;:GOTO 290
420 IF FAST=150 THEN BEEP: GOTO 290 ELSE FAST=FAST
    +1:LOCATE 1,68:PRINT FAST;:GOTO 290
430 ON VAL(A$) GOTO 520,1080,470,500,480,
    450,410,420,3040
440 '------ SET SPEED ---------------------------
450 IF SP=15 THEN BEEP ELSE SP=SP+1:LOCATE 1,46:PR
    INT SP
460 GOTO 290
470 STA=NUM:LOCATE 1,20:PRINT STA:GOTO 290
480 IF SP=-15 THEN BEEP ELSE SP=SP-1:LOCATE 1,46:P
    RINT SP
490 GOTO 290
500 EN=NUM:LOCATE 1,31:PRINT EN:GOTO 290
510 '----- ANIMATE PICTURES ----------------------
520 LOCATE 17,1:PRINT"PRESS SPACE BAR TO PAUSE MOV
    EMENT":LOCATE 18,1:PRINT"USE ARROW KEYS FOR SP
    EED":IF STA>EN THEN QQ=-1 ELSE QQ=1
530 FOR I=STA TO EN STEP QQ
540 PL1=PL:IF PL+SP<1 THEN PL=570 ELSE IF PL+SP >
    580 THEN PL=1
550 PL=PL+SP
560 LOCATE 1,8:PRINT I:ON I GOTO 630,640,650,660,670,
    680,690,700,710,720,730,740,750,760,770,780 ,790,
    800,810,820
570 FOR N=0 TO FAST*4:NEXT:A$=INKEY$:IF A$=" " THE
    N 2340 ELSE IF A$<>"" AND LEN(A$)<>2 THEN LOCA
    TE 1,8:PRINT NUM:LOCATE 17,1:PRINT STRING$
    (150,32):GOTO 290
580 IF LEN(A$)<>2 THEN 620
590 C=ASC(RIGHT$(A$,1)):IF C=77 THEN SP=SP+1 ELSE
    IF C=75 THEN SP=SP-1
600 IF SP=-16 THEN SP=-15 ELSE IF SP=16 THEN SP=15
610 LOCATE 1,46:PRINT SP
620 NEXT:GOTO 530
630 PUT(PL1,52),U%,PSET:PUT(PL,52),A%:GOTO 570
640 PUT(PL1,52),U%,PSET:PUT(PL,52),B%:GOTO 570
650 PUT(PL1,52),U%,PSET:PUT(PL,52),C%:GOTO 570
660 PUT(PL1,52),U%,PSET:PUT(PL,52),D%:GOTO 570
670 PUT(PL1,52),U%,PSET:PUT(PL,52),E%:GOTO 570
680 PUT(PL1,52),U%,PSET:PUT(PL,52),F%:GOTO 570
690 PUT(PL1,52),U%,PSET:PUT(PL,52),G%:GOTO 570
700 PUT(PL1,52),U%,PSET:PUT(PL,52),H%:GOTO 570
710 PUT(PL1,52),U%,PSET:PUT(PL,52),I%:GOTO 570
720 PUT(PL1,52),U%,PSET:PUT(PL,52),J%:GOTO 570
730 PUT(PL1,52),U%,PSET:PUT(PL,52),K%:GOTO 570
740 PUT(PL1,52),U%,PSET:PUT(PL,52),L%:GOTO 570
750 PUT(PL1,52),U%,PSET:PUT(PL,52),M%:GOTO 570
760 PUT(PL1,52),U%,PSET:PUT(PL,52),N%:GOTO 570
770 PUT(PL1,52),U%,PSET:PUT(PL,52),O%:GOTO 570
780 PUT(PL1,52),U%,PSET:PUT(PL,52),P%:GOTO 570
790 PUT(PL1,52),U%,PSET:PUT(PL,52),Q%:GOTO 570
800 PUT(PL1,52),U%,PSET:PUT(PL,52),R%:GOTO 570
810 PUT(PL1,52),U%,PSET:PUT(PL,52),S%:GOTO 570
820 PUT(PL1,52),U%,PSET:PUT(PL,52),T%:GOTO 570
830 C=ASC(RIGHT$(A$,1)):IF C=77 THEN NUM=NUM+1 ELS
    E IF C=75 THEN NUM=NUM-1
840 IF NUM=0 THEN NUM=20 ELSE IF NUM=21 THEN NUM=1
850 LOCATE 1,8:PRINT NUM:GOTO 290
860 '------- SAVE PICTURE -----------------------
870 CLS:ON ERROR GOTO 25010
880 LOCATE 10,10:PRINT"F-FILES   A-ABORT SAVE  AN
    Y OTHER KEY TO CONTINUE"
890 A$=INKEY$:IF A$="" THEN 890 ELSE IF A$="F" THE
    N FILES"*.ANI" ELSE IF A$="A" THEN GOSUB 2310:
    GOTO 190
900 PRINT:PRINT:PRINT:INPUT"NAME OF FILE TO SAVE";
    A$:IF A$="" THEN GOSUB 2310:GOTO 190
910 IF INSTR(A$,".")<>0 THEN CLS:LOCATE 9,10:PRINT
    "NO EXTENSION PLEASE..":GOTO 880
920 IF LEN(A$)>8 THEN CLS:LOCATE 9,10:PRINT"NO MOR
    E THAN 8 CHARACTERS PLEASE":GOTO 880
930 IF VAL(RIGHT$(A$,1))>0  OR RIGHT$(A$,1)="0" TH
    EN CLS:LOCATE 9,10:PRINT"THE FIRST CHARACTER
                               CAN'T BE A NUMBER..":GOTO 880
940 GOSUB 2310
950 A$=A$+".ANI"
960 DEF SEG=&HB800:BSAVE A$,0,&H4000:PRINT"IT HAS
    BEEN SAVED.    PRESS ANY KEY TO CONTINUE":PR
    INT:PRINT:PRINT
970 A$=INKEY$:IF A$="" THEN 970 ELSE CLS:GOSUB
    2310: GOTO 190
980 '------- LOAD PICTURE -----------------------
990 CLS:ON ERROR GOTO 25000
1000 LOCATE 10,10:PRINT"F-FILES  A-ABORT LOAD  ANY
     KEY OTHER TO CONTINUE"
1010 A$=INKEY$:IF A$="" THEN 1010 ELSE IF A$="F" T
     HEN FILES"*.ANI" ELSE IF A$="A" THEN GOSUB
     2310:GOTO 190
1020 PRINT:PRINT:PRINT:INPUT"NAME OF FILE TO LOAD"
     ;A$:IF A$="" THEN GOSUB 2310:GOTO 190
1030 IF INSTR(A$,".")<>0 THEN CLS:LOCATE 9,10:PRIN
     T"NO EXTENSION PLEASE..":GOTO 1000
1040 IF LEN(A$)>8 THEN CLS:LOCATE 9,10:PRINT"NO MO
     RE THAN 8 CHARACTERS PLEASE":GOTO 1000
1050 IF VAL(RIGHT$(A$,1))>0 OR RIGHT$(A$,1)="0" TH
     EN CLS:LOCATE 9,10:PRINT"THE FIRST CHARACTER
     CAN'T BE A NUMBER..":GOTO 1000
1060 A$=A$+".ANI":DEF SEG=&HB800:BLOAD A$,0:ON ERR
     OR GOTO 0:GOTO 190
1080 LOCATE 16,1:PRINT"TYPE IN 21 TO ABORT":PRINT"
     NUMBER SET AT THE TOP OF THE SCREEN IS PIC. T
     O READ FROM RETURN FOR SAME"
1085 LOCATE 18,1:INPUT"EDIT PICTURE NUMBER";B:IF B
```

```
     <0 OR B>21 THEN BEEP:GOTO 1080
1090 IF B=21 THEN CLS:GOSUB 2310:GOTO 190
1100 IF B=0 THEN B=NUM
1110 '------- PUT PICTURE TO EDIT ON SCREEN ------
1120 CLS:LOCATE 1,24:PRINT"WAIT...":ON NUM GOTO
     1130,1140,1150,1160,1170,1180,1190,1200,1210,
     1220,1230,1240,1250,1260,1270,1280,1290,
     1300,1310,1320
1130 PUT(1,50),A%:GOTO 1340
1140 PUT(1,50),B%:GOTO 1340
1150 PUT(1,50),C%:GOTO 1340
1160 PUT(1,50),D%:GOTO 1340
1170 PUT(1,50),E%:GOTO 1340
1180 PUT(1,50),F%:GOTO 1340
1190 PUT(1,50),G%:GOTO 1340
1200 PUT(1,50),H%:GOTO 1340
1210 PUT(1,50),I%:GOTO 1340
1220 PUT(1,50),J%:GOTO 1340
1230 PUT(1,50),K%:GOTO 1340
1240 PUT(1,50),L%:GOTO 1340
1250 PUT(1,50),M%:GOTO 1340
1260 PUT(1,50),N%:GOTO 1340
1270 PUT(1,50),O%:GOTO 1340
1280 PUT(1,50),P%:GOTO 1340
1290 PUT(1,50),Q%:GOTO 1340
1300 PUT(1,50),R%:GOTO 1340
1310 PUT(1,50),S%:GOTO 1340
1320 PUT(1,50),T%:GOTO 1340
1330 '------- GET ON-OFF POINTS ------------------
1340 FOR I=1 TO 20:FOR X=1 TO 54:A(I,X)=POINT(X,I+49)
1350 NEXT:NEXT
1360 '------- DRAW EDITING SCREEN ----------------
1370 FOR I=1 TO 20:LOCATE 3+I,14:PRINT "..........
     ............................................";:FO
     R J=1 TO 54:IF A(I,J)=1 THEN LOCATE 3+I,
     13+J:PRINT"#"
1380 NEXT:NEXT
1390 LOCATE 1,30:PRINT"(Q)UIT    (D)RAW    (M)OVE
     (E)RASE   (C)LEAR    (S)AVE    (I)NVERSE"
1400 GOTO 1550
1410 '------ PLACE CURSOR ------------------------
1420 BLINK%=(BLINK%+1) MOD 20:IF BLINK%<10
     THEN 1470 ELSE 1440
1430 '------ CURSOR OFF --------------------------
1440 IF A(ROW,COLUMN)=0 THEN CH$="." ELSE IF A(ROW
     ,COLUMN)=1 THEN CH$="#"
1450 GOTO 1480
1460 '------- CURSOR ON --------------------------
1470 IF CURS=-1 THEN CH$="-" ELSE IF CURS=0 THEN C
     H$="*" ELSE IF CURS=1 THEN CH$="+"
1480 LOCATE 3+ROW,13+COLUMN:PRINT CH$;:RETURN
1490 '------- REMOVE CURSOR ----------------------
1500 IF A(ROW,COLUMN)=0 THEN CH$="." ELSE IF A(ROW
     ,COLUMN)=1 THEN CH$="#"
1510 LOCATE 3+ROW,13+COLUMN:PRINT CH$;:RETURN
1520 LOCATE 24,18:PRINT"wait";:FOR I=1 TO 20:LOCAT
     E 3+I,14:PRINT STRING$(54,46);
1530 NEXT:ERASE A:DIM A(20,54):LOCATE 24,18:PRINT"
     ";:PUT(1,50),U%,PSET:RETURN
1540 '-------- SET CURSOR ------------------------
1550 ROW=1:COLUMN=1:CURS=0
1560 '-------- MAIN LOOP -------------------------
1570 BLINK%=0:IF CURS=-1 THEN A(ROW,COLUMN)=0:PSET
     (COLUMN,ROW+49),0 ELSE IF CURS=+1 THEN A(ROW,
     COLUMN)=1:PSET(COLUMN,ROW+49),1
1580 GOSUB 1420
1590 A$=INKEY$:DEF SEG:POKE 106,0:IF LEN(A$)=0 THE
     N 1580 ELSE IF LEN(A$)=1 THEN 1600 ELSE IF LE
     N(A$)=2 THEN 1720
1600 CODE1=ASC(A$) AND &H5F
1610 '----- READ KEYS ---------------------------
1620 IF CODE1=ASC("E") THEN 2040
1630 IF CODE1=ASC("M") THEN 2050
1640 IF CODE1=ASC("D") THEN 2060
1650 IF CODE1=ASC("C") THEN 2080
1660 IF CODE1=ASC("S") THEN 2100
1670 IF CODE1=ASC("Q") THEN GOSUB 2310:GOTO 190
1680 IF CODE1=ASC("I") THEN 1710
1690 GOTO 1580
1700 '------- INVERSE A PICTURE ------------------
1710 GET(1,50)-(54,70),U%:PUT(1,50),U%,PRESET:GET(
     1,75)-(54,95),U%:GOTO 1340
1720 IF ASC(A$)<>0 THEN 1570 ELSE CODE2=ASC(RIGHT$
     (A$,1)):GOSUB 1500
1730 '-------- READ ARROW KEYS -------------------
1740 IF CODE2=71 THEN 1840
1750 IF CODE2=73 THEN 1870
1760 IF CODE2=79 THEN 1900
1770 IF CODE2=81 THEN 1930
1780 IF CODE2=72 THEN 1960
1790 IF CODE2=75 THEN 1980
1800 IF CODE2=77 THEN 2000
1810 IF CODE2=80 THEN 2020
1820 GOTO 1580
1830 '------- MOVE THE CURSOR --------------------
1840 IF ROW=1 THEN ROW=21
1850 IF COLUMN=1 THEN COLUMN=55
1860 ROW=ROW-1:COLUMN=COLUMN-1:GOTO 1570
1870 IF ROW=1 THEN ROW=21
1880 IF COLUMN=54 THEN COLUMN=0
1890 ROW=ROW-1:COLUMN=COLUMN+1:GOTO 1570
1900 IF ROW=20 THEN ROW=0
1910 IF COLUMN=1 THEN COLUMN=55
```

```
1920 ROW=ROW+1:COLUMN=COLUMN-1:GOTO 1570
1930 IF ROW=20 THEN ROW=0
1940 IF COLUMN=54 THEN COLUMN=0
1950 ROW=ROW+1:COLUMN=COLUMN+1:GOTO 1570
1960 IF ROW=1 THEN ROW=21
1970 ROW=ROW-1:GOTO 1570
1980 IF COLUMN=1 THEN COLUMN=55
1990 COLUMN=COLUMN-1:GOTO 1570
2000 IF COLUMN=54 THEN COLUMN=0
2010 COLUMN=COLUMN+1:GOTO 1570
2020 IF ROW=20 THEN ROW=0
2030 ROW=ROW+1:GOTO 1570
2040 CURS=-1:GOTO 1570
2050 CURS=0:GOTO 1570
2060 CURS=+1:GOTO 1570
2070 '------- CLEAR THE PICTURE -------------
2080 GOSUB 1520:GOTO 1550
2090 '----------- SAVE THE PICTURE ----------
2100 LOCATE 1,24:PRINT"WAIT...":ON B GOTO 2110,
     2120,2130,2140,2150,2160,2170,2180,2190,2200,
     2210,2220,2230,2240,2250,2260,2270,2280,2290,2300
2110 GET(1,50)-(54,70),A%:GOSUB 2310:GOTO 190
2120 GET(1,50)-(54,70),B%:GOSUB 2310:GOTO 190
2130 GET(1,50)-(54,70),C%:GOSUB 2310:GOTO 190
2140 GET(1,50)-(54,70),D%:GOSUB 2310:GOTO 190
2150 GET(1,50)-(54,70),E%:GOSUB 2310:GOTO 190
2160 GET(1,50)-(54,70),F%:GOSUB 2310:GOTO 190
2170 GET(1,50)-(54,70),G%:GOSUB 2310:GOTO 190
2180 GET(1,50)-(54,70),H%:GOSUB 2310:GOTO 190
2190 GET(1,50)-(54,70),I%:GOSUB 2310:GOTO 190
2200 GET(1,50)-(54,70),J%:GOSUB 2310:GOTO 190
2210 GET(1,50)-(54,70),K%:GOSUB 2310:GOTO 190
2220 GET(1,50)-(54,70),L%:GOSUB 2310:GOTO 190
2230 GET(1,50)-(54,70),M%:GOSUB 2310:GOTO 190
2240 GET(1,50)-(54,70),N%:GOSUB 2310:GOTO 190
2250 GET(1,50)-(54,70),O%:GOSUB 2310:GOTO 190
2260 GET(1,50)-(54,70),P%:GOSUB 2310:GOTO 190
2270 GET(1,50)-(54,70),Q%:GOSUB 2310:GOTO 190
2280 GET(1,50)-(54,70),R%:GOSUB 2310:GOTO 190
2290 GET(1,50)-(54,70),S%:GOSUB 2310:GOTO 190
2300 GET(1,50)-(54,70),T%:GOSUB 2310:GOT 190
2310 CLS:PUT(1,10),A%:PUT(61,10),B%:PUT(121,10),C%
     :PUT(181,10),D%:PUT(241,10),E%:PUT(301,10),F%
     :PUT(361,10),G%:PUT(421,10),H%:PUT(481,10),I%
2315 PUT(541,10),J%
2320 PUT(1,81),K%:PUT(61,81),L%:PUT(121,81),M%:PUT
     (181,81),N%:PUT(241,81),O%:PUT(301,81),P%:PUT
     (361,81),Q%:PUT(421,81),R%:PUT(481,81),S%
2325 PUT(541,81),T%,PSET
2330 RETURN
2340 IF INKEY$="" THEN 2340 ELSE 620
2350 '----------- INSERT A BLANK PICTURE ---------
2360 LOCATE 18,1:PRINT"ARE YOU SURE? THIS WILL MOVE
     20 OFF THE END"
2370 A$=INKEY$:IF A$="" THEN 2370 ELSE IF A$<>"Y"
     THEN LOCATE 18,1:PRINT STRING$ (50,32):GOTO 190
2380 ON NUM GOTO 2390,2400,2410,2420,2430,2440,2450,
     2460,2470,2480,2490,2500,2510,2520,2530,2540,
     2550,2560,2570,2820
2390 GET(1,10)-(54,30),B%
2400 GET(61,10)-(114,30),C%
2410 GET(121,10)-(174,30),D%
2420 GET(181,10)-(234,30),E%
2430 GET(241,10)-(294,30),F%
2440 GET(301,10)-(354,30),G%
2450 GET(361,10)-(414,30),H%
2460 GET(421,10)-(474,30),I%
2470 GET(481,10)-(534,30),J%
2480 GET(541,10)-(594,30),K%
2490 GET(1,81)-(54,101),L%
2500 GET(61,81)-(114,101),M%
2510 GET(121,81)-(174,101),N%
2520 GET(181,81)-(234,101),O%
2530 GET(241,81)-(294,101),P%
2540 GET(301,81)-(354,101),Q%
2550 GET(361,81)-(414,101),R%
2560 GET(421,81)-(474,101),S%
2570 GET(481,81)-(534,101),T%
2580 CLS:ON NUM GOTO 2590,2600,2610,2620,2630,2640
     ,2650,2650,2660,2670,2680,2690,2700,2710,2720
     ,2730,2740,2750,2760,2770,2780
2590 GET(1,50)-(54,70),A%:GOSUB 2310:GOTO 190
2600 GET(1,50)-(54,70),B%:GOSUB 2310:GOTO 190
2610 GET(1,50)-(54,70),C%:GOSUB 2310:GOTO 190
2620 GET(1,50)-(54,70),D%:GOSUB 2310:GOTO 190
2630 GET(1,50)-(54,70),E%:GOSUB 2310:GOTO 190
2640 GET(1,50)-(54,70),F%:GOSUB 2310:GOTO 190
2650 GET(1,50)-(54,70),G%:GOSUB 2310:GOTO 190
2660 GET(1,50)-(54,70),H%:GOSUB 2310:GOTO 190
2670 GET(1,50)-(54,70),I%:GOSUB 2310:GOTO 190
2680 GET(1,50)-(54,70),J%:GOSUB 2310:GOTO 190
2690 GET(1,50)-(54,70),K%:GOSUB 2310:GOTO 190
2700 GET(1,50)-(54,70),L%:GOSUB 2310:GOTO 190
2710 GET(1,50)-(54,70),M%:GOSUB 2310:GOTO 190
2720 GET(1,50)-(54,70),N%:GOSUB 2310:GOTO 190
2730 GET(1,50)-(54,70),O%:GOSUB 2310:GOTO 190
2740 GET(1,50)-(54,70),P%:GOSUB 2310:GOTO 190
2750 GET(1,50)-(54,70),Q%:GOSUB 2310:GOTO 190
2760 GET(1,50)-(54,70),R%:GOSUB 2310:GOTO 190
2770 GET(1,50)-(54,70),S%:GOSUB 2310:GOTO 190
2780 GET(1,50)-(54,70),T%:GOSUB 2310:GOT 190
2790 '-------- DELETE A PICTURE ------------------
2800 LOCATE 18,1:PRINT"ARE YOU SURE WANT TO DELETE
     THIS NUMBER"
```

```
2810 A$=INKEY$:IF A$="" THEN 2810 ELSE IF A$ <>"Y"
     THEN LOCATE 18,1:PRINT STRING$(50,32):GOTO 190
2820 ON NUM GOTO 2830,2840,2850,2860,2870,2880,2890
     ,2900,2910,2920,2930,2940,2950,2960,2970,2980,
     2990,3000,3010,3020
2830 GET(61,10)-(114,30),A%
2840 GET(121,10)-(174,30),B%
2850 GET(181,10)-(234,30),C%
2860 GET(241,10)-(294,30),D%
2870 GET(301,10)-(354,30),E%
2880 GET(361,10)-(414,30),F%
2890 GET(421,10)-(474,30),G%
2900 GET(481,10)-(534,30),H%
2910 GET(541,10)-(594,30),I%
2920 GET(1,81)-(54,101),J%
2930 GET(61,81)-(114,101),K%
2940 GET(121,81)-(174,101),L%
2950 GET(181,81)-(234,101),M%
2960 GET(241,81)-(294,101),N%
2970 GET(301,81)-(354,101),O%
2980 GET(361,81)-(414,101),P%
2990 GET(421,81)-(474,101),Q%
3000 GET(481,81)-(534,101),R%
3010 GET(541,81)-(594,101),S%
3020 CLS:PUT(1,10)-(54,30),T%:GOSUB 2310:GOTO 190
3030 '-------- INVERSE ALL THE PICTURE ------------
3040 CLS:PUT(1,10),A%,PRESET:PUT(61,10),B%,PRESET:P
     UT(121,10),C%,PRESET:PUT(181,10),D%,PRESET:PUT
     (241,10),E%,PRESET
3045 PUT(301,81),F%,PRESET:PUT(361,10),G%,PRESET:PU
     T(421,10),H%,PRESET:PUT(481,10),I%,PRESET:PUT(
     541,10),J%,PRESET
3050 PUT(1,81),K%,PRESET:PUT(61,81),L%,PRESET:PUT
     (121,81),M%,PRESET:PUT(181,81), N%,PRESET:PUT(241,
     81),O%,PRESET
3055 PUT(301,81),P%,PRESET:PUT(361,81),Q%,PRESET:PU
     T(421,81),R%,PRESET:PUT(481,81),S%,PRESET:PUT(
     541,81),T%,PRESET
3060 GOTO 190
3070 '------- MAKE A PROGRAM ---------------------
3080 LOCATE 18,1:PRINT"ARE YOU SURE YOU WANT TO MAK
     E THIS SET-UP INTO A PROGRAM?"
3090 A$=INKEY$:IF A$="" THEN 3090 ELSE IF A$="Y" TH
     EN 3100 ELSE LOCATE 18,1:PRINT STRING$(66,32):
     GOTO 290
3100 G=30:OPEN :O:,#1,"PRG.BAS"
3110 PRINT #1,"10 CLS:KEY OFF:SCREEN 2:SP="+STR$(SP
     )+":PL=1"
3120 IF STA>EN THEN QQ=-1 ELSE QQ=1
3130 A$="20 DIM ":FOR I=STA TO EN STEP QQ:IF I<> ST
     A THEN A$=A$+","
3140 ON I GOTO 3150,3160,3170,3180,3190,3200,3210,
     3220,3230,3240,3250,3260,3270, 3280,3290,3300,
     3310,3320,3330,3340
3150 A$=A$+"A%(144)":GOTO 3350
3160 A$=A$+"B%(144)":GOTO 3350
3170 A$=A$+"C%(144)":GOTO 3350
3180 A$=A$+"D%(144)":GOTO 3350
3190 A$=A$+"E%(144)":GOTO 3350
3200 A$=A$+"F%(144)":GOTO 3350
3210 A$=A$+"G%(144)":GOTO 3350
3220 A$=A$+"H%(144)":GOTO 3350
3230 A$=A$+"I%(144)":GOTO 3350
3240 A$=A$+"J%(144)":GOTO 3350
3250 A$=A$+"K%(144)":GOTO 3350
3260 A$=A$+"L%(144)":GOTO 3350
3270 A$=A$+"M%(144)":GOTO 3350
3280 A$=A$+"N%(144)":GOTO 3350
3290 A$=A$+"O%(144)":GOTO 3350
3300 A$=A$+"P%(144)":GOTO 3350
3310 A$=A$+"Q%(144)":GOTO 3350
3320 A$=A$+"R%(144)":GOTO 3350
3330 A$=A$+"S%(144)":GOTO 3350
3340 A$=A$+"T%(144)":GOTO 3350
3350 NEXT
3355 '------ -------------------------------------
     ---------------
3360 A$=A$+",U%(144)":PRINT #1,A$:PRINT #1,"30 GET(
     1,1)-(54,20),U%"
3370 IF STA>EN THEN QQ=-1 ELSE QQ=1
3380 FOR P=STA TO EN STEP QQ
3390 CLS:ON P GOTO 3400,3410,3420,3430,3440, 3450,
     3460,3470,3480,3490,3500,3510,3 520,3530,3540,
     3550,3560,3570,3580,3590
3400 PUT(1,50),A%:GOTO 3600
3410 PUT(1,50),B%:GOTO 3600
3420 PUT(1,50),C%:GOTO 3600
3430 PUT(1,50),D%:GOTO 3600
3440 PUT(1,50),E%:GOTO 3600
3450 PUT(1,50),F%:GOTO 3600
3460 PUT(1,50),G%:GOTO 3600
3470 PUT(1,50),H%:GOTO 3600
3480 PUT(1,50),I%:GOTO 3600
3490 PUT(1,50),J%:GOTO 3600
3500 PUT(1,50),K%:GOTO 3600
3510 PUT(1,50),L%:GOTO 3600
3520 PUT(1,50),M%:GOTO 3600
3530 PUT(1,50),M%:GOTO 3600
3540 PUT(1,50),O%:GOTO 3600
3550 PUT(1,50),P%:GOTO 3600
3560 PUT(1,50),Q%:GOTO 3600
3570 PUT(1,50),R%:GOTO 3600
3580 PUT(1,50),S%:GOTO 3600
3590 PUT(1,50),T%:GOTO 3600
```

```
3600 FOR X=1 TO 54:G=G+10:A$=STR$(G):A$=RIGHT$(A$,L
     EN(A$)-1):FOR I=1 TO 20
3605 IF POINT(X,I+49)=1  THEN A$=A$+":PSET(":B$=STR
     $(X):A$=A$+RIGHT$(B$,LEN(B$)-1):A$=A$+",":B$=S
     TR$(I+49):A$=A$+RIGHT$(B$,LEN(B$)-1:A$=A$+")"
3610 NEXT:IF LEN(A$)>6 THEN B$=LEFT$(A$,LEN(STR$(G)
     )-1):A$=RIGHT$(A$,(LEN(A$)-LEN(B$))-1):A$=B$+"
     "+A$:PRINT #1,A$ ELSE G=G-10
3620 NEXT
3630 G=G+10:A$=STR$(G):A$=RIGHT$(A$,LEN(A$)-1):ON P
     GOTO 3640,3650,3670,3680,3690,3700,3710,3720,
     3730,3740,3750,3760,3770,3780, 3790,3800,
     3810,3820,3830
3640 A$=A$+" GET(1,50)-(54,70),A%":GOTO 3840
3650 A$=A$+" GET(1,50)-(54,70),B%":GOTO 3840
3660 A$=A$+" GET(1,50)-(54,70),C%":GOTO 3840
3670 A$=A$+" GET(1,50)-(54,70),D%":GOTO 3840
3680 A$=A$+" GET(1,50)-(54,70),E%":GOTO 3840
3690 A$=A$+" GET(1,50)-(54,70),F%":GOTO 3840
3700 A$=A$+" GET(1,50)-(54,70),G%":GOTO 3840
3710 A$=A$+" GET(1,50)-(54,70),H%":GOTO 3840
3720 A$=A$+" GET(1,50)-(54,70),I%":GOTO 3840
3730 A$=A$+" GET(1,50)-(54,70),J%":GOTO 3840
3740 A$=A$+" GET(1,50)-(54,70),K%":GOTO 3840
3750 A$=A$+" GET(1,50)-(54,70),L%":GOTO 3840
3760 A$=A$+" GET(1,50)-(54,70),M%":GOTO 3840
3770 A$=A$+" GET(1,50)-(54,70),N%":GOTO 3840
3780 A$=A$+" GET(1,50)-(54,70),O%":GOTO 3840
3790 A$=A$+" GET(1,50)-(54,70),P%":GOTO 3840
3800 A$=A$+" GET(1,50)-(54,70),Q%":GOTO 3840
3810 A$=A$+" GET(1,50)-(54,70),R%":GOTO 3840
3820 A$=A$+" GET(1,50)-(54,70),S%":GOTO 3840
3830 A$=A$+" GET(1,50)-(54,70),T%":GOTO 3840
3840 A$=A$+":CLS":PRINT #1,A$:NEXT
3850 IF STA=EN THEN STN=1:EA=1:GOTO 3880
3860 IF STA>EN THEN STN=STA-EN:EA=1 ELSE EA=EN-STA:
     STN=1
3870 IF EA+QQ=0 THEN EA=EA-QQ
3880 G=G+10:R=G:A$=RIGHT$(STR$(G),LEN(STR$(G))-1)+"
     FOR I="+RIGHT$(STR$(STN),LEN(STR$(STN))-1)+" T
     O"+STR$(EA+QQ)+" STEP"+STR$(QQ):PRINT
     #1,A$:G=G+10
3890 A$=RIGHT$(STR$(G),LEN(STR$(G))-1)+" FOR P=1 TO
     ":A$=A$+STR$(INT(FAST*4.2))+":NEXT:PL1=PL:IF P
     L+SP<1 THEN PL=570 ELSE IF PL+SP>580 THEN PL=1"
3900 G=G+10:PRINT #1,A$
3910 A$=RIGHT$(STR$(G),LEN(STR$(G))-1)+" PL=PL+SP":
     PRINT #1,A$:G=G+10
3915 A$=RIGHT$(STR$(G),LEN(STR$(G))-1)+" ON I GOTO"
3920 X=G+10:A$=A$+STR$(X):FOR I=STA TO EN+(QQ*-1) S
     TEP QQ:X=X+10:A$=A$+","+RIGHT$(STR$(X),LEN(STR
     $(X))-1):NEXT:PRINT#1,A$
3930 IF STA>EN THEN Q=STA:W=EN ELSE IF EN>STA THEN
     Q=EN:W=STA
3940 FOR P=W TO Q
3950 G=G+10:A$=RIGHT$(STR$(G),LEN(STR$(G))-1):ON P
     GOTO 3960,3970,3980,3990,4000,4010,4020,4030,
     4040,4050,4060,4070,4080,4090,4100,4110,4120,
     4130,4140,4150
3960 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),A%:":
     GOTO 4160
3970 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),B%:":
     GOTO 4160
3980 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),C%:":
     GOTO 4160
3990 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),D%:":
     GOTO 4160
4000 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),E%:":
     GOTO 4160
4010 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),F%:":
     GOTO 4160
4020 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),G%:":
     GOTO 4160
4030 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),H%:":
     GOTO 4160
4040 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),I%:":
     GOTO 4160
4050 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),J%:":
     GOTO 4160
4060 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),K%:":
     GOTO 4160
4070 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),L%:":
     GOTO 4160
4080 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),M%:":
     GOTO 4160
4090 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),N%:":
     GOTO 4160
4100 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),O%:":
     GOTO 4160
4110 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),P%:":
     GOTO 4160
4120 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),Q%:":
     GOTO 4160
4130 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),R%:":
     GOTO 4160
4140 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),S%:":
     GOTO 4160
4150 A$=A$+" PUT(PL1,52),U%,PSET:PUT(PL,52),T%:":
     GOTO 4160
4160 A$=A$+"GOTO"+STR$(X+10):PRINT #1,A$:NEXT:G=X+10
4165 A$=RIGHT$(STR$(G),LEN(STR$(G))-1)
4170 A$=RIGHT$(STR$(G),LEN(STR$(G))-1):A$=A$+" NE
     XT:GOTO"+STR$(R):PRINT #1,A$
4180 CLOSE #1:CLS:PRINT"BEFORE YOU DO ANYTHING EL
```

```
    SE LOAD THE PROGRAM <PRG> AND THEN SAVE IT U
    NDER THE NAME YOU WANT":END
25000 GOSUB 25020:RESUME 990
25010 GOSUB 25020:RESUME 870
25020 PRINT:PRINT "                    DISK ERR
    OR #"ERR
25030 PRINT:PRINT "                    HIT SPAC
    E BAR TO CONTINUE"
25040 A$=INKEY$:IF EN 25040 ELSE END
```

사운드

베이직은 다양한 사운드명령을 갖고 있지만, 오퍼레이팅시스템(OS : Operating System)에서는 단지 "Beep"음만이 가능하다. 따라서 Beep이외의 다른 사운드를 내기 위해서는 8253 타이머칩을 직접 프로그램해야 한다. 8253칩의 채널2는 컴퓨터의 스피커에 연결되어 있다. 이 채널2가 모드3으로 프로그램되면 주어진 주파수에 의해 구형파를 만든다.

이렇게 만들어진 구형파로 표현되는 사운드는 단순하기 때문에 그 가장자리를 매끄럽게 하여 좀더 경쾌한 사운드가 나는 정현파(Sine Wave)로 줄여나간다. 그러나 불행히도 8253칩은 파형의 진폭은 바꿀 수 없으므로 소리의 볼륨에 대한 제어를 할 수 없다.

스피커는 사운드를 발생하기 위해 한 입력이 아니라 두 입력을 받아들인다. 즉 타이머칩과 8255 페리페럴 인터페이스에서 스피커로 신호를 보내는데, 두 칩중 어느 한 칩에서 펄스레이트(Pulse Rate)는 변하게 되고, 두 칩의 동작을 합성하면 특수한 사운드효과를 낼 수 있게 된다.

톤의 형성

컴퓨터가 아무 일도 하지 않는 동안에 어떻게 소리를 내는지와 다른 동작이 진행되는 동안에 어떻게 소리를 내는지에 대해서 알아 보자. 후자의 방법으로 소리를 발생시키기 위해서는 CPU를 독립적으로 운용하는 8253 타이머칩의 프로그래밍 신호가 필요하다.

여기에 예를 든 방법은 CPU가 직접 스피커를 제어해야 하기 때문에 소프트웨어로 타이머칩의 하드웨어 작업을 처리해야만 하는데 이러한 기법은 좀더 다양한 제어를 스피커에 가할 수 있어 대부분 스피커의 특수 음향효과는 이 기법을 사용한다.

(1) 베이직의 경우

베이직의 SOUND문을 이용하면 폭넓은 범위의 주파수 및 지속기간에 걸쳐 톤을 발생시킬 수 있다. 주파수는 초당 사이클(37~32767)로 주어지며, 지속기간은 0~65535까지의 펄스들로 카운트되는데 여기서는 초당 18.2의 펄스가 존재한다. 예를 들어 SOUND 440, 91은 5초(5×18.2) 동안 동조음표(Tuning Note)를 발생시킨다.

중간 C에서 시작하는 옥타브(Octave)의 주파수들은 다음과 같다.

중간 C	523.3
D	587.3
E	659.3
F	698.5
G	784.0
A	880.0
B	987.7

한 옥타브 더 높은 주파수들은 전 옥타브의 주파수 값에 비해 거의 2배이며, 두 옥타브 더 높은 주파수들은 다시 한 옥타브 높은 주파수 값보다 2배의 값을 갖는다. 반대로 한 옥타브 낮은 주파수들은 전 값의 반정도이다.

사운드발생기(Sound Generator)는 SOUND 주파수, 지속기간, 볼륨, 채널의 형태를 갖는다. 여기서 볼륨은 0~15까지 지정이 가능하며 디폴트치는 8이다. 채널번호는 0~2까지 지정이 가능하며 보통 0으로 디폴트되어 있다. 낮은 볼륨에서 D소화음(Minor Chord)를 발생시키기 위해서는 다음과 같이 쓰기를 한다.

```
100 SOUND ON            ;멀티채널사운드 가능
110 SOUND 587,50,3,0 ;D화음발생
120 SOUND 699,50,3,1 ;F화음발생
130 SOUND 880,50,3,2 ;A화음발생
```

(2) 어셈블리어의 경우

8255 페리페럴 인터페이스 어댑터로부터 사운드를 발생시키기 위해서는 원하는 주파수에서 스피커(비트 1)에 연결된 포트 B의 비트를 켜거나 끄거나 하기만 하면 된다. 포트 B는 포트 어드레스 61H에 위치한다(비록 AT는 8255 페리페럴 인터페이스와 같은 어드레스 위치는 아니지만 똑같은 포트 어드레스와 비트 할당을 한다).

포트 B의 비트 0는 스피커에 연결된 순서대로 타이머칩 채널2의 게이트를 제어하며 포트 B의 비트 1은 차례대로 게이트를 제어하기 위해서 타이머 채널과 OFF동작으로 분리한다. 그림 1은 사운드 주파수를 어떻게 세트하는지를 보여준다.

다음의 예에서는 FREQUENCY와 NUMBER-CYCLES이라는 두 변수가 있다. "FREQUENCY"로 명명된 한 변수는 ON / OFF동작간에 존재하

는 지연루프의 카운터로 사용된다. 변수의 번호가 작으면 작을수록 ON / OFF 전환은 빨라지고, 주파수는 점점 높아진다. 반면에 "NUMBER-CYCLES"는 톤의 지속기간을 설정한다. 변수의 번호가 작을수록 톤의 지속기간은 길어진다.

하드웨어 인터럽트는 이 루틴동안 클리어(동작하지 않음)된다. 왜냐하면 타이머 인터럽트가 사운드를 들을 수 있게 톤을 변조하는 만큼 주파수나 레귤레러티(Regularity : 초당 18.2번)를 발생하기 때문이다. 그러나 인터럽트가 동작하지 않을 때마다 BIOS시간 카운트는 늦어진다. 따라서 시간읽기가 연속적으로 이루어질 경우, 적당한 조정이 이루어지지 않으면 혼돈을 일쉽다(리스트 1 참조).

다른 동작과 동시에 톤을 발생

8253 타이머칩은 8088 CPU와 독립적으로 운용되기 때문에 다른 운용이 계속되는 동안 계속해서 사운드를 발생시키는 것은 쉬운 일이다.

특별한 주파수를 발생시키려면 단지 8253 타이머칩의 채널 2를 프로그램하면 된다. 그리고 나중에 그 사운드를 중지시키려면 이 타이머칩을 재프로그램해야 한다.

(1) 베이직의 경우

베이직에서 SOUND문은 동시에 사운드를 발생할 수는 없지만, PLAY문은 만일 동시에 사운드를 발생하도록 특별한 명령을 받는다면 사운드를 발생할 수 있다.

스트링(String)이 문자 MB(Music Background)를 갖게될 경우, 그 스트링은 특수한 버퍼에 저장되어 다른 프로그램 운용이 진행되는 동안에도 수행된다. 반면에 MF(Music Foreground) 문자를 갖게 되면 스트링이 끝날 때까지 모든 다른 프로그램운용을 중지시킨다. 여기서 싱글 톤 A는 배경(Background)으로 연주된다.

```
100 PLAY "MB A"  'A를 연주...
110 ...          '연주중...
```

MB모드일 때 X=PLAY(0)문은 연주가 계속 유지되도록 음표의 번호들을 32까지 되돌린다.

(2) 어셈블리어의 경우

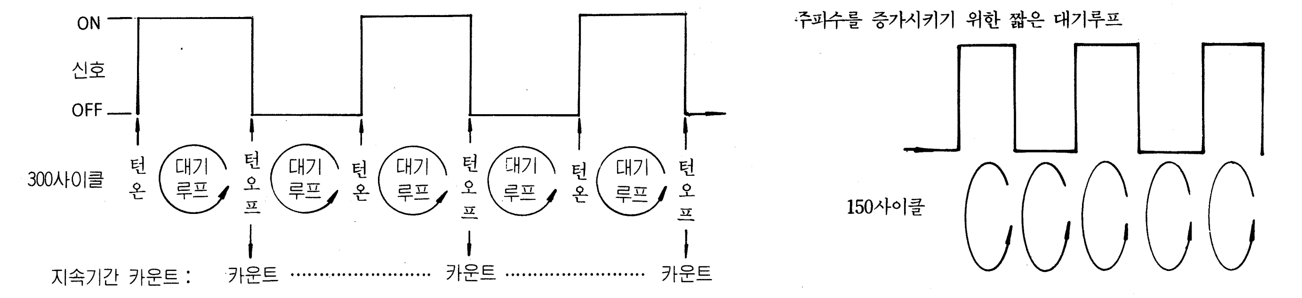
채널 2에 카운터를 간단하게 하기 위해서 우선 타이머칩은 8255페리페럴 인터페이스(61H에서)의 포트 B를 통해 동작되어야 한다. 또한 1.19M를 원하는 초당 사이클의 번호로 나눔으로서 래치에 대한 카운터를 계산한다.

사운드는 채널 2에 대한 게이트가 끊어질 때까지 계속되기 때문에 포트 B의 비트 1을 0으로 리세트시켜 게이트를 끊어야 한다. 그렇지 않으면 사운

〔리스트 1〕

```
NUMBER_CYCLES EQU  1000
FREQUENCY     EQU  300
PORT_B        EQU  61H
              CLI                      ;클리어 되어 인터럽트 동작 안함
              MOV  DX,NUMBER_CYCLES    ;DX에 톤의 음량을 카운트
              IN   AL,PORT_B           ;포트B를 AL에 입력
              AND  AL,11111110B        ;타이머칩으로부터 스피커분리
NEXT_CYCLE:   OR   AL,00000010B        ;스피커를 ON
              OUT  PORT_B,AL           ;포트B에 명령지시
              MOV  CX,FREQUENCY        ;CX에 1/2사이클동안 지연값 전송
FIRST_HALF:   LOOP FIRST_HALF          ;스피커가 ON일 동안 지연형성
              AMD  AL,11111101B        ;스피커 OFF
              OUT  PORT_B,AL           ;포트B로 명령지시
              MOV  CX,FREQUENCY        ;2번째1/2사이클동안 지연값 CX로 옮김
SECOND_HALF:  LOOP SECOND_HALF         ;스피커 OFF동안 지연형성
              DEC  DX                  ;사이클번호에서 1을 뺀다
              JNZ  NEXT_CYCLE          ;만일 0이면 지속기간 고갈됨
              STI                      ;인터럽트 다시 동작 시킴
```

〔그림 1〕 8255칩에 의해 발생되는 사운드



드는 무한히 계속된다.

톤의 지속기간을 정확하게 지정하기 위해서는 지난 3월호에서 설명했던 BIOS 시간카운트를 사용한다. 다음의 예에서는 피치(Pitch)를 초당 440 사이클로 설정하였고 지연은 랜덤하게 키를 치는 동안 대기하였다가 제공된다(리스트 2 참조).

〔리스트 2〕

```
;---8255칩의 포트B를 세팅함으로서 채널2를 인에이블:
  PORT_B       EQU  61H          ;8255칩상의 PB의 어드레스설정
               IN   AL,PORT_B    ;포트B 입력
               OR   AL,3         ;2개의 하위비트 ON(3=00000011B)
               OUT  PORT_B,AL    ;포트B로 변화된 바이트 출력
;---I/O레지스터 셋업:
  COMMAND_REG  EQU  43H          ;명령레지스터의 어드레스설정
  CHANNEL_2    EQU  42H          ;채널2의 어드레스설정
               MOV  AL,10110110B ;2바이트 채널2에 대한 비트형태 전송
               OUT  COMMAND_REG,AL;모드3에서의 바이너리번호 출력
;---래치로 카운터출력:
               MOV  AX,2705      ;카운터(1190000/440)를 전송
               OUT  CHANNEL_2,AL ;LSB출력
               MOV  AL,AH        ;AL를 통해 출력해야하기 때문에
                                  MSB AL로 이동
               OUT  CHANNEL_2,AL ;MSB출력
;---키를 치는동안 대기 지연:
               MOV  AH,1         ;INT 21H의 기능번호 설정
               INT  21H          ;인터럽트 호출
;---사운드를 끈다:
               IN   AL,PORT_B    ;포트B의 바이트를 AL로 입력
               AND  AL,11111100B ;2개의 하위비트를 반드시
                                  0으로 세트
               OUT  PORT B,AL    ;포트B로 변화된 바이트출력
```

스피커의 비프

어떤 프로그램에서는 다양한 음으로 경고를 위한 "Beep"를 발생하기를 원하는 경우가 있다. 그러나 이 비프(Beep) 는 베이직에서는 쉽게 만들어지지만 오퍼레이팅시스템에서는 그와 같은 비프기능을 전혀 제공하지 않기 때문에 간접적으로 시스템을 스타트업할 때 들리는 비프소리를 액세스하도록 한다.

톤을 바꾸기 위해서는 전체의 사운드발생루틴이 어셈블리어로 프로그램되어야 한다. 프로그램의 메시지에 따라 비프를 조절하기 위해서는 약간 상상력이 필요하다. 즉 다가올 운명을 예언하기 위해서는 슬라이딩 톤(Sliding Tone)으로 사이렌을 울리게 한다든지, 프린터를 온라인상태로 하고 컴퓨터 스피커와 프린터 스피커(프린터 데이터라인에서 출력 ASCII 7)간을 서로 교체하는 등의 예를 들 수 있다.

(1) 베이직의 경우

베이직에서는 간단하게 "Beep"를 쓸 수 있다. 여기서는 적당한 에러로 비프(Beep)와 물음표를 얻어 본다.

```
100 INPUT"Enter your age",AGE  '나이를 입력
110 IF AGE>100 THEN BEEP:PRINT
"Are you really over 100?"     '에러?
                               '등등....
```

그러면 다른 주파수의 지속기간의 비트(Beat)를 위해 SOUND문을 사용해 본다. 그 형식은 SOUND 피치(Pitch), 지속기간(Duration)이며, 여기서 피치는 초당 사이클(3000은 중간범위)로 주어지고 지속기간은 1/18초의 간격으로 주어진다. 예를 들어 SOUND 3000, 18은 약 1초동안 중간범위의 소리를 낸다. 이 경우에 스피커는 높은 음과 낮은 음사이를 위협을 느낄정도로 급속하게 변한다.

```
100 FOR N=1 TO 200 '교환가능번호설정
110 SOUND 500,1    '1/18초에 대한 저음
120 SOUND 5000,1   '1/18초에 대한 고음
130 NEXT           '반복
```

(2) 어셈블리어의 경우

앞에서도 언급했듯이 오퍼레이팅시스템은 특별하게 사운드를 위한 기능을 제공하지는 않는다. 그러나 ASCII의 캐릭터 7을 DOS나 BIOS기능중 하나를 표준 디바이스로 사용하여 쓰기를 함으로서-즉, 비디오 모니터로 ASCII 캐릭터 7을 보낸다-간단하게 우리 귀에 익숙한 "Beep"사운드를 발생시킬 수 있다. ASCII 7은 "벨" 제어코드로 해석되며, 그의 심볼은 스크린상으로 표현되지 않는다. DOS 인터럽트 21H의 기능 2가 가장 쉽다.

```
MOV AH,2 ;스크린상의 캐릭터를 쓰기 위한 기능
MOV DL,7 ;ASCII 7를 DL레지스터로 전송
INT 21H  ;스피커가 Beep소리를 냄
```

INT 10H의 BIOS기능 AH에서는 ASCII 7을 처리할 때 Beep를 발생시키지 않는다. 대신에 캐릭터를 디스플레이한다.

단순하게 Beep를 발생시키기 위해서는 8255 페리페럴 인터페이스 칩에 이용한 방법이 가장 간단하다. 다음은 컴퓨터의 스위치를 켰을 때 BIOS의 비프 톤이 거칠게 반복되는 경우이다. (리스트 3 참조).

〔리스트 3〕

```
;---스피커의 Beep:
            MOV  DX,840     ;사이클의 번호를 카운트
            IN   AL,61H     ;8255칩상의 포트B를 읽어들임
            AND  AL,0FEH    ;8253타이머비트를 OFF
NEXTCYCLE:  OR   AL,2       ;스피커를 ON
            OUT  61H,AL     ;역으로 포트B로 바이트 출력
            MOV  CX,150     ;파형의 1/2를 지속기간으로 설정
CYCLEUP:    LOOP CYCLEUP    ;신호가 높은동안 지연
            AND  AL,0FDH    ;스피커비트를 OFF
            OUT  61H,AL     ;포트B로 바이트를 출력
CYCLEDOWN:  LOOP CYCLEDOWN  ;신호가 낮은동안 지연
            DEC  DX         ;사이클의 번호로 부터 1감소
            JNZ  NEXTCYCLE  ;DX가 0이면, 다음 사이클로 점프
```

톤의 스트링 구성

여기서는 컴퓨터가 아무일도 하지 않는 동안에 사운드의 스트링이 어떻게 수행되는지 알아 본다. 사운드가 비동시성일 때 스트링은 멜로디나 음향효과의 디스플레이중 하나겠지만 사운드가 동시성일 때는 멜로디의 디스플레이는 가능하나 음향효과의 디스플레이는 불가능하다.

사운드 스트링은 베이직에 의해 제공되는 우수한 형태이다. 그러나 어셈블리어의 스크래치(Scratch)로부터 스트링을 구성하는데에는 많은 양의 작업이 필요하다. 사운드를 발생시키기 위해서는 앞에서 설명한 톤을 형성하는 방법과 다른 동작을 하면서 동시에 톤을 구성하는 방법중 하나를 사용할 수 있다.

두 방법 모두 한 톤을 스타팅하고 그것을 타이밍하고, 또 다음 톤을 스타팅하는 동작을 계속 반복하여 모든 사운드의 스트링을 형성하는데 그중 한 방법은 연속되는 톤들의 주파수를 유지하고 다른 한 방법은 각각에 대한 지속기간(다른 길이의 톤을 제공하는 것이 필요하다)을 유지한다.

따라서 이러한 지속기간은 BIOS 시간카운트를 사용하여 측정하는 것이 가능하다.

(1) 베이직의 경우

PLAY문은 베이직의 가장 우수한 형태중의 하나이다. 이 문은 음표(Note) 들이 어떻게 연주되는지에 관한 정보를 갖고 있는 음표의 스트링으로 구성되어 있다. 음표들은 문자 A~G로 쓰며, 샵(Sharp)이나 플랫(Flats) 기호가 뒤따른다. 샵은 #나 +로 나타내며 플랫(임시기호)은 －로 나타낸다.

PLAY "CC01/01/80"와 PLAY "CD-D"는 등가이다.(그러나 비검정키의 음표를 보이기 위해서 임시기호를 사용하지는 않는다.) 음표들을 명명하는 두번째 방법은 0에서 84까지의 코드번호를 계산하는 것인데, 여기서 0은 리세트와 같고 1~84는 제일 아래음(Bottom)부터 시작해서 7개 옥타브에서 84개의 가능한 음표들과 일치한다.

예를 들어 문자 N : PLAY "N3N72N44" 를 갖는 번호의 음표를 실행해 보자. 옥타브 범위는 7까지 허용되며, 각 옥타브는 C에서 B음까지 가능하다. 그리고 각 옥타브들은 0에서 6까지 지정되어 있다. 즉 중간 C의 경우는 옥타브 3에서 시작한다. 또 현재의 옥타브는 O(문자 "O"는 0이 아님) 다음에 옥타브번호를 삽입하므로써 스트링내의 임의 점에서 바뀔 수 있으며, 그 옥타브 뒤에 이어지는 모든 음표들은 그 옥타브내에서 다른 옥타브 세팅이 이루어질 때까지 연주된다. 아무 것도 초기에 세트가 되지 않았을 때에는 옥타브 4가 사용된다. 뿐만 아니라 PLAY "O3CO4CO5CO6C"으로 점점 C를 높여서 연주할 수 있다.

옥타브를 바꾸기 위한 또 다른 방법은 스트링내에 〉나 〈기호를 삽입시킴으로서 기능한데, 이러한 전환기호들은 각각 한 옥타브를 올리거나 내리는 작용을 한다. 따라서 PLAY "O3C〉C〉C〉C"과 같은 명령도 역시 점차적으로 C를 높여서 연주한다.

음표들은 코드번호를 삽입하기 이전에 문자 L를 삽입함으로서 다른 길이를 제공하기도 한다. 다음의 모든 음표들은 다른 길이의 코드가 나타날 때까지 그 길이로 주어진다. 코드는 0에서 64까지의 번호를 가지는데, 여기서 1은 전음표(Whole Note)이고 64는 64분 음표이다. 4분 음표를 만들려면 L4를 쓰면 된다.

음표들이 연주되는 곳에서 템포(Tempo) 는 문자 T 다음에 분당 4분음표의 번호로 주어지는 32부터 255까지 번호가 계속 되는 템포코드에 의해 세트된다. 문자가 지정되지 않았을 때의 음표길이는 L4로 디폴트되며, 120은 템포를 나타낸다.

단, 단음표의 길이만을 변화시키려면 문자 L없이 음표 다음에 길이 값을 놓는다. 예를 들어 PLAY "L4CDE16FC"는 16분음표로서 E를 연주하고 나머지 모두는 4분음표로 연주한다.

쉼표들도 음표길이가 카운트되는 것과 같은 방법으로 카운트된다. 문자 P("Pause"에 대한) 다음에 1에서 64까지의 번호를 놓는다. P1은 전음표 늘임표로 주어지고, P64는 64분음표 늘임표로 주어진다. 음표 다음에 마침표를 놓는 것은 보통의 음악표기에서와 같은 효과를 갖는다. 즉 그 마침표로 인해 음표의 길이는 반음(Half)정도 연장된다. 제2 마침표는 거기에서 다시 반음정도의 길이를 연장한다.

디폴트에 의해 음표들은 그들의 특

정 지속기간의 7/8분 동안 연주된다. 그 음표들을 전체의 지속기간(음을 끊지 않고) 동안 연주하기 위해서 ML을 스트링에 놓는다. 그들을 3/4분 지속기간(스타카토 : Staccato-음을 명확히 끊는 것)동안 연주하려면 MS를 스트링에 놓는다. 그리고 원래상태로 되돌리기 위해서는 MN을 쓴다.

통상적으로 모든 다른 프로그램동작은 스트링이 완료될 때까지 중지되어 있다. PLAY문 다음에 오는 문이 실행되는 동안 스트링을 배경에서 연주되도록 하려면 MB를 사용한다. 그리고 일반적인 환경으로 복원하려면 MF를 쓴다.

최종적으로 PLAY문는 꽤 많은 양의 스트링내에서 연주될 수 있도록 서브스트링을 허용한다. 이것은 스트링의 한 부분이 스트링 변수로 셋업될 수 있다는 것을 의미한다. 그리고 그 변수는 PLAY문을 구성하는 스트링내에서 호출될 수 있다.

예를 들어 S$=EEEEE"이면 PLAY "CDXS$;FG"문에서 음표E는 5번 반복한다. 여기서 문자 X다음에 변수명이 오고 그 다음에 세미콜론(;)이 온다는 점에 주의한다. (컴파일된 프로그램에서는 VARPTR$을 사용하는 또 다른 방법이 필요하다.)

이번 예에서는 익숙한 대형 괘종시계의 차임벨을 연주한다. 스트링은 우선 레가토(Legato : 끊이지 않고 부드럽고 미끄럽게)에서 연주하는 멜로디를 세트하고 그 다음 템포를 세트하고서는 옥타브를 시작하고 그리고 마지막으로 4분음표, 늘임표, 역으로 같은 4개의 음표들을 연주한다. 코드간의 공간은 전적으로 프로그래머의 편의를 위한 것이다. 베이직에서는 이 공간은 무시한다.

```
100 PLAY "ME T40 O3 ECD<G P32 G>DEC"
```

(2) 어셈블리어의 경우

여기서는 사운드를 발생시키기 위해 8253 타이머칩을 사용한다. 단지 8개의 음표를 연주할 수 있지만 조그만 수정을 가하면 매우 다양한 음표를 만들 수 있다.

여기에 3개의 데이터 스트링이 있다. 첫번째 스트링은 임의의 지연 마침표(임의의 마침표를 변화시키는 것은 템포를 변화시킨다)의 배수로서 각 음표의 지속기간을 세트한다. 두번째 스트링은 각각 8개 음표의 주파수를 유지하는데 그 값들은 8253 칩의 채널 2의 래치 레지스터내에 위치할 때 요구되는 톤에 기인하는 것이다. 세번째 스트링은 1에서 8까지 8개의 주파수에 상당하는 코드번호의 형식으로 멜로디를 유지한다. 이 스트링은 종료를 알리는 FF로 끝난다(**리스트 4** 참조).

루틴은 단지 멜로디의 각각의 음표를 읽고 일치하는 주파수를 찾고 그 주파수를 채널 2에 놓는다. 그 음표에 할당된 지속기간은 시간카운트를 사용하는 지연루프로 공급되며, 지연이 끝나면 다음 음표가 진행된다. 그림 2가 이러한 루틴을 나타낸 것이다.

다른 동작과 동시에 톤의 스트링 구성

다른 운용과 동시에 톤의 스트링을 구성하는 것은 비록 베이직에서는 쉽지만 리얼타임 프로그래밍에서는 다루기 힘든 비트이나 8255을 기반으로한 사운드 발생방법을 사용하여 음향효과를 가하지 않는 순수 음악적인 톤의 스트링은 동시에 연주될 수 있다.

(1) 베이직의 경우

다른 운용과 동시에 사운드를 발생하는 톤 스트링은 베이직의 매우 정교한 PLAY문의 또 다른 옵션이다.

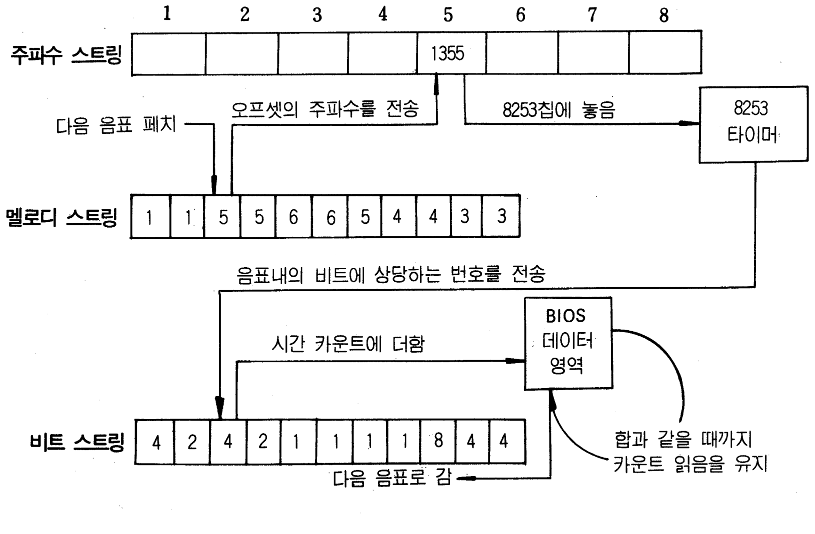
간단히 MB를 제어스트링의 시작에 추가한다. 이것은 "Music Background"를 나타낸다. 그리고 PLAY문으로 하여금 멜로디가 끝날 때까지 다른 프로그램 운용을 멈추고 원래대로 복원하도록 하기 위해 MF를 삽입한다. 이 예는 박스를 그리고 채워 넣는 동안(여기서는 그래픽 능력을 요구한다.) 한 스케일을 연주한다.

```
100 PLAY "ME T100 O3 L4;CDEFG>ABC"
    '중간C음으로부터 한 스케일 연주
110 LINE(10,10)-(80,80),1,BF
    '똑같은 시간에 박스를 그림
```

(2) 어셈블리어의 경우

여기서 설명할 루틴은 앞절에서 보인 비리얼타임 루틴보다 좀더 정교한

〔그림 2〕 음표의 스트링을 연주



〔리스트 4〕

```
;---데이터 세그먼트:
BEAT        DB  10,9,8,7,6,5,4,3,2  ;각음표의 지속기간
FREQUENCY   DW  2280,2031,1809,1709 ;주파수의 일람표
            DW  1521,1355,1207,1139
MELODY      DB  1,2,3,4,5,6,7,8,0FFH;각 음표의 주파수 코드
;---초기화:
PORT_B      EQU 61H
COMMAND_REG EQU 43H
LATCH2      EQU 42H
            IN  AL,PORT_B           ;포트B의 현재상태입력
            OR  AL,00000011B        ;스피커와 타이머를 채널2에
                                      인에이블
            OUT PORT_B,AL           ;바이트를 교체
            MOV SI,0                ;멜로디/비트 스트링의
                                      번지초기화
            MOV AL,0B6H             ;모드3에 대한 채널2 초기화
            OUT COMMAND_REG,AL      ;명령레지스터로 바이트전송
;---음율을 찾고 여기서 얻은 주파수를 채널2에 놓는다:
NEXT_NOTE:  LEA BX,MELODY           ;멜로디스트링의 오프셋을 로드
            MOV AL,[BX][SI]         ;7번째 음율스트링에 대한 코드
                                      전송
            CMP AL,0FFH             ;FF인가 체크(제작자가 스트링을
                                      끝낼 때까지)
            JE  NO_MORE             ;FF면 루틴의 끝으로 점프
            CBW                     ;AL을 워드길이오퍼렌드로 바꿈
       ;주파수를 얻음:
            MOV BX.OFFSET FREQUENCY ;주파수 일람표의 오프셋 전송
            DEC AX                  ;0부터 카운팅을 시작하여 1감소
            SHL AX,1                ;워드길이 일람표에 의해 AX를
                                      1만큼 왼쪽으로 시프트
            MOV DI,AX               ;어드레싱을 위해 DI로 전송
            MOV DX,[BX][DI]         ;일람표로부터 주파수를 찾음
       ;음율연주 시작:
            MOV AL,DL               ;주파수의 하위바이트 전송준비
            OUT LATCH2,AL           ;레치레지스터로
                                     출력(I/O레지스터를 통해)
            MOV AL,DH               ;상위바이트 준비
            OUT LATCH2,AL           ;상위바이트 출력
;---지연루프를 만듬:
            MOV AH,0                ;BIOS시간카운트 실행위한 기능
            INT 1AH                 ;카운트실행
            MOV BX,OFFSET BEAT      ;비트스트링의 오프셋 전송
            MOV CL,[BX][SI]         ;SI번호 음율에 대한 비트값 전송
            MOV CH, 0               ;워드로 사용하기 위해 CX의
                                      반을 클리어
            MOV BX,DX               ;AX로부터 BIOS카운트의 하위워드
                                      전송
            ADD BX,CX               ;현재의 BIOS카운트에 비트카운트
                                      더함
STILL_SOUND:INT 1AH                 ;카운트 실행
            CMP DX,BX               ;음율의 마지막 카운트를
                                     기준으로 카운트 체크
            JNE STILL_SOUND         ;같으면 계속해서 소리발생
            INC SI                  ;아니면 다음 음율 지시
            JMP NEXT_NOTE           ;다음 음율 실행
;---끝마친다:
NO_MORE:    IN  AL,PORT_B           ;포트B에 바이트입력
            AND AL,0FCH             ;스피커 비트를 끈다
            OUT 61H,AL              ;포트B에 바이트를 교체
```

것이다. 이것은 먼저 타이머 인터럽트가 어떻게 재프로그램되는지에 대한 이해를 요구한다. 그 루틴은 인터럽트 벡터에 의해 지시되며, BIOS시간카운트가 새롭게 됨과 동시에 초당 18.2번 수행된다. 보통 몇 라인만 실제로 실행되고 (이 몇라인으로도 소리의 변화가 필요없는지에 대한 결정을 하는데 충분하다) CPU가 다른 작업을 자유롭게 할 수 있도록 루틴을 되돌린다.

BIOS시간카운트는 각 음표의 지속시간을 측정하는데 사용된다. 한 음표에서 다른 음표변화가 이루어질 때 새 음표의 지속시간은 BIOS시간카운트의 펄스번호로서 계산되고 그 값은 현재 카운트를 읽어서 더한 값이다. 시간값은 루틴을 불러낼 때마다 체크되고, 대기 값이 마지막으로 나타날 때 사건의 한 체인은 다음 음표를 조사하고, 그 주파수를 8253칩의 채널 2로 프로그램하여 새로운 지속기간 카운터를 셋업한다. 기타 코드는 스트링의 처음과 최종음표가 특수한 경우에 필요하다(리스트 5 참조).

슬라이딩 톤의 구성

슬라이딩(Sliding)톤은 주파수를 연속적으로 변화시킴으로서 만들어진다. 베이직과 어셈블리어 프로그래밍 모두 이 톤을 얻을 수 있다. 이러한 음향효과는 톤의 각 세그먼트의 지속기간을 약간 짧게하거나, 톤이 감소함에 따라 그 지속기간을 약간 길게 함으로서 좀 더 드라마틱(Dramatic)하게 얻을 수 있다.

(1) 베이직의 경우

다음의 예는 베이직에서 톤에 대한 지속기간을 매우 짧게하여 루프내에 SOUND명령을 간단히 놓고 카운터가 통과할 때마다 임의 배수로 주파수를 증가시킨 것이다.

```
100 FOR N=1 TO 500 STEP 15
110 SOUND 400+N,1
120 NEXT
```

〔리스트 5〕

```
;---데이터 세그먼트:
BEAT         DB   10,9,8,7,6,5,4,3,2        ;각 음의 지속기간
FREQUENCY    DW   2280,2031,1809,1709       ;주파수의 일람표
             DW   1521,1355,1207,1139       ;
MELODY       DB   1,2,3,4,5,6,7,8,0FFH      ;각 음의 주파수코드
HOLDIP       DW   0                         ;원래의 INT 1CH벡터를 저장
HOLDCS       DW   0                         ;상동
SOUND_NOW?   DB   1                         ;소리의 ON/OFF 체크
FIRST_NOTE?  DB   1                         ;특별히 1번째 음표의 경우 체크
END_NOTE     DW   0                         ;종료 음표에서 타이머 카운트 유지
WHICH_NOTE   DW   0                         ;스트링내의 현재음표에 번지지정
;---인터럽트 벡터 초기화
   ;벡터변화:
             PUSH DS                        ;DS를 스택에 넣음
             MOV  AX,SEG MELODY2            ;루틴의 세그먼트 설정
             MOV  DS,AX                     ;DS로 전송
             MOV  DX,OFFSET MELODY2         ;루틴의 오프셋 설정
             MOV  AL,1CH                    ;인터럽트 벡터 교체
             MOV  AH,25H                    ;베터를 세트하기 위한 기능
             INT  21H                       ;벡터 바꿈
             POP  DS                        ;DS로 복원

;
;---프로그램만 전송,임의 시간에서 소리루틴 호출
;

;---프로그램 종료시,원래 벡터로 재배치:
             MOV  DX,0FF53H                 ;DX에 원래의INT 1C 오프셋 설정
             MOV  AX,0F000H                 ;DS에 원래의 INT 1C세그먼트 설정
             MOV  DS,AX                     ;
             MOV  AL,1CH                    ;인터럽트 번호
             MOV  AH,25H                    ;인터럽트벡터를 변화시키는 기능
             INT  21H                       ;원래의 인터럽트를 교체
             RET
;---인터럽트:
MELODY2      PROC FAR                       ;변경된 레지스터 저장
             PUSH AH                        ;
             PUSH BX                        ;
             PUSH CX                        ;
             PUSH DX                        ;
             PUSH DI                        ;
             PUSH SI                        ;
             PUSH DS                        ;
             MOV  AX,SS:[114]               ;스택에 있는 원래의 DS를 전송
             MOV  DS,AX                     ;DS복원
             CMP  SOUND_NOW?,1              ;현재 요구되는 소리가 어떤것인지 체크
             JE   PLAY_IT                   ;맞으면 연주
             JMP  NOT_NOW                   ;아니면 인터럽트를 점프
PLAY_IT:     CMP  FIRST_NOTE?,0             ;스트링을 시작하는지 체크
             JE   TIME_CHECK                ;입하면 타이밍루프로 점프
                                            ;그렇지 않으면 멜로디 스트링을 시작
;---초기화:
PORT_B       EQU  61H                       ;포트명이 같은 것으로 설정
COMMAND_REQ  EQU  43H                       ;
LATCH2       EQU  42H                       ;
             IN   AL,PORT_B                 ;포트B의 현재상태 입력
             OR   AL,00000011B              ;스피커나 타이머를 채널2에 인에이블
             OUT  PORT_,AL                  ;바이트를 교체
             MOV  SI,0                      ;멜로디/비트 스트링 초기화
             MOV  AL,0B6H                   ;모드3에 대해 채널2를 초기화
             OUT  COMMAND_REG,AL            ;명령레지스터로 바이트출력
             MOV  FIRST_NOTE?,0             ;현재 진행되고 있는 멜로디플래그 초기화
;---음율을 찾아 주파수를 설정, 채널2에 놓음:
NEXT_NOTE:   LEA  BX,MELODY                 ;멜로디 스트링의 오프셋 로드
             MOV  SI,WHICH_NOTE             ;현재음율로 지시되는 것을 SI에 전송
             MOV  AL,[BX][SI]               ;스트링의 N번째 음율에 대한 코드 설정
             CMP  AL,0FFH                   ;FF와 비교 (제작자가 스트링을 완료할
                                                  때까지)
             JE   NO_MORE                   ;FF면 종료루틴으로 점프
             CBW                            ;워드길이 오퍼랜드로 AL를 변환
   ;주파수를 얻음:
             MOV  BX,OFFSET FREQUENCY       ;주파수 일람표의 오프셋 설정
             DEC  AX                        ;0부터 카운트가 시작되도록 AX에서 1을
                                                                 뺀다
             SHL  AX,1                      ;워드길이 일람표이므로 AX를 왼쪽으로
                                              시프트
             MOV  DI,AX                     ;어드레싱을 하기 위해 DI로 이동
             MOV  DX,[BX][DI]               ;일람표로부터 주파수를 전송
   ;음율연주시작:
             MOV  AL,DL                     ;주파수의 하위바이트를 전송 준비
             OUT  LATCH2,AL                 ;레치레지스터로 출력(I/O레지스터를
                                                           통해)
             MOV  AL,DH                     ;상위바이트 준비
             OUT  LATCH2,AL                 ;상위바이트 출력
;---지연루프 만듬:
TIME_IT:     MOV  AH,0                      ;BIOS시간카운트 전송 위한 기능
             INT  1AH                       ;카운트를 실행
             MOV  BX,OFFSET BEAT            ;비트 스트링의 오프셋 전송
             MOV  CL,[BX][SI]               ;SI음표번호에 대한 비트값 전송
             MOV  CH,0                      ;워드로 사용하기 위해 CX높은 음의 반을
                                                       클리어
             MOV  BX,DX                     ;DX로부터 BIOS카운트의 하위워드 전송
             ADD  BX,CX                     ;현재BIOS카운트에 펄스카운트를 더함
             MOV  END_NOTE,BX               ;그 값을 종료음표에 저장
TIME_CHECK:  MOV  AH,0                      ;BIOS시간카운트를 실행하기 위한 기능
             INT  1AH                       ;카운트 실행
             CMP  DX,END_NOTE               ;종료음표 카운트와 실행 카운트 비교
             JNE  NOT_NOW                   ;같지 않으면 인터럽트를 멈춤
             MOV  SI,WHICH_NOTE             ;아니면 다음 음율을 시작
             INC  SI                        ;하나씩 음율 카운트를 증가 실행
             MOV  WHICH_NOTE,SI             ;음율 카운트를 저장
```

```
        JMP  NEXT_NOTE           ;다음 음율을 시작
;---루틴 끝냄:
NO_MORE:  IN   AL,PORT_B         ;포트B의 바이트 입력
        AND  AL,0FCH             ;스피커 비트를 끈다
        OUT  61H,AL              ;포트B의 바이트를 바꿈
        MOV  SOUND_NOW?,0        ;변하기 쉬운 하나의 스트링 연주를
                                   OFF로 설정
        MOV  FIRST_NOTE?,1       ;다음 음율 시작
NOT_NOW:  POP  DS                ;변환된 레지스터 재저장
        POP  SI                  ;
        POP  DI                  ;
        POP  DX                  ;
        POP  CX                  ;
        POP  BX                  ;
        POP  AX                  ;
        IRET                     ;인터럽트로부터 되돌아감
MELODY2   ENDP
```

(2) 어셈블리어의 경우

어셈블리어의 경우에는 8255 페리페럴 인터페이스 칩으로 제어되는 사운드발생 방법을 사용하는 것이 가장 쉽다. 톤을 발생하는 빈 타이밍 루프를 사용하여 포트 B의 비트 1을 1과 0사이에서 변조한다. 그리고 타이밍 루프가 CX에 한 값을 설정하면 복원될 때마다 그 값이 약간 증가하거나 감소한다(리스트 6 참조).

이러한 단순한 방법은 높은 범위가 낮은 범위에서 보다 상당히 빠르게 통과되는데에 의한 것이다. 그리고 이러한 효과는 지속기간이 짧은 간격에서 유용하다. 그 지속기간이 짧지 않으면 톤이 증가함에 따라 DX는 재로드될 때(예의 6번째 라인) 좀더 높은 값이 주어지도록 코드가 추가되어야 한다.

〔리스트 6〕

```
;---타이머 칩을 디스에이블
PB        EQU  61H          ;PB를 8255포트B의 어드레스와 같게 설정
          IN   AL,PB        ;PB에 바이트 입력
          OR   AL,1         ;비트0를 OFF
          OUT  PB,AL        ;역으로 PB의 바뀐 바이트 출력
;---사운드의 주파수와 지속기간 설정
          MOV. BX,9000      ;카운트값 초기화,아래로 감소
          MOV  DX,3000      ;소리는 3000사이클동안 계속됨
REPEAT:                     ;각 사이클후 여기로 되돌아옴
;---스피커 비트를 켬
          OR   AL,00000010B;비트1을 "ON"으로 가함
          OUT  PB,AL        ;PB내의 바이트를 ON으로 함
          MOV  CX,BX        ;1번째 사이클의 반에 대한 카운터를 설정
CYCLE1:   LOOP CYCLE1       ;1000번 반복하는 동안 루프에서 헛돈다
;---스피커 비트를 끈다
          ADD  AL,11111101B;비트1을 "OFF"로 가함
          OUT  PB,AL        ;PB내의 바이트를 OFF로 설정
          MOV  CX,BX        ;2번째 사이클의 반에 대한 카운터를 설정
CYCLE2:   LOOP CYCLE2       ;1000번 반복하는 동안 루프에서 헛돈다
;---다음 사이클로 간다
          DEC  BX           ;카운트 감소,주파수 증가시킴
          DEC  BX           ;다시 위와 동일하게 실행
          DEC  DX           ;남은 지속기간 감소
          JNZ  REPEAT       ;DX가 0이 아니면 또 다른 사이클로 점프
                            ;아니면 소리를 멈춤
```

음향효과의 구성

음향효과는 일반적으로 톤의 주파수에 연속된 변화를 가져온다. 그러나 일반적으로 음향효과가 다른 프로그램운용과 동시에 발생되도록 하는 것은 쉽지 않다.

(1) 베이직의 경우

SOUND문과 PLAY문이 베이직에서 워낙 강력하기 때문에 정교한 음향효과를 쉽게 발생시킬 수 있다. 그러나 정교한 음악효과의 정도는 모든 음향효과가 순수한 음악적인 톤으로 구성되어야 할 경우 음향 왜곡의 영향은 톤을 매우 빨리 변하게 함으로서 사람의 귀가 이에 둔감하도록 하는 것을 의미한다.

예를 들어 날카롭게 "떨리는 목소리"는 여러 옥타브들을 분리하여 설정된 같은 톤간에서 신속하게 이리저리 스위칭됨으로서 발생된다.

```
100 FOR N=1 TO 100 '지속기간 설정
110 PLAY"l64t255"  '가능한 가장 빠른 템포
120 PLAY"O1A"      '낮은 A 연주
130 PLAY"O5A"      '높은 A 연주
140 NEXT           '반복한다
```

단지 초당 수 사이클에 걸쳐 범위가 변한다면, 그 음향효과의 결과는 비브라토(Vibrato : 소리를 떨어서 내는 효과)의 종류가 된다.

```
100 FOR N=1 TO 100 '지속기간 설정
110 SOUND 440,1    'A를 연주
120 SOUND 445,1    '약간 높이 올려 연주
130 NEXT           '반복
```

(2) 어셈블리어의 경우

어셈블리 프로그래밍은 스피커가 OFF된 동안의 간격이 ON된 동안의 간격과 같지 않으면 매끄럽지 않은 톤을 발생시킨다. 이러한 대칭성에 의한 왜곡은 버징(Buzzing : 윙윙거리는 소리)과 클리킹(Clicking : 딸깍거리는 소리)등과 같은 다양한 소리를 발생시킨다.

버징은 두 간격의 차가 50에서 1일때 발생한다. 그 차가 10에서 20배일때 버즈는 개개의 클리킹 소리로 감소한다. 두가지 경우에서 사운드는 톤 발생에서 보여준 기본적인 기법을 사용하여 8255 페리페럴 인터페이스 칩으로부터 발생되어야 한다.

다음의 예가 이러한 버즈에 대한 것이다(리스트 7 참조).

〔리스트 7〕

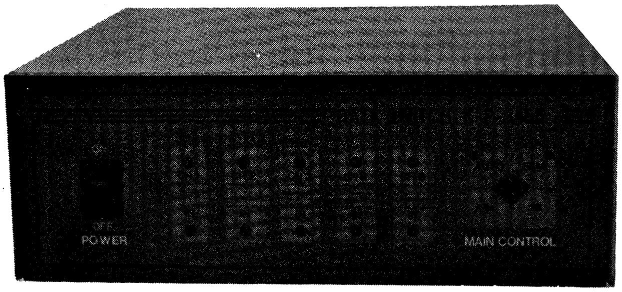
```
NUMBER_CYCLES  EQU  300              ;시간스피커의 번호 ON/OFF
FREQUENCY1     EQU  50               ;시간 ON
FREQUENCY2     EQU  3200             ;시간 OFF
PORT_B         EQU  61H              ;8255칩의 포트B 어드레스
               CLI                   ;인터럽트를 디스에이블
               MOV  DX,NUMBER_CYCLES ;톤의 길이를 DX에 카운트
               IN   AL,11111110B     ;포트B를 AL에 입력
NEXT_CYCLE:    OR   AL,00000010B     ;타이머 칩으로부터 스피커를 분리
               OUT  PORT_B,AL        ;포트B로 코멘드를 출력
               MOV  CX,FREQUENCY1    ;CX로 1/2사이클에 대한 지연 전송
FRIST_HALF:    LOOP FIRST_HALF       ;스피커가 ON일 동안 지연됨
               AND  AL,11111101B     ;스피커를 OFF
               OUT  PORT_B,AL        ;포트B로 코멘드를 출력
               MOV  CX,FREQUENCY2    ;2번째 사이클의 반동안의 지연을 전송
SECOND_HALF:   LOOP SECOND_HALF      ;스피커가 OFF일 동안 지연됨
               DEC  DX               ;사이클의 번호로 부터 1을 뺀다
               JNZ  NEXT_CYCLE       ;0이면 지속기간은 소모된다
               STI                   ;인터럽트를 다시 가능하도록 설정한다
```

클리킹소리를 발생시킬 때도 똑같은 코드를 사용한다. 그러나 FREQUENCY_2의 값을 약 40000까지 바꾼다.

PRINTER 주변기기 전문 메이커

- 컴퓨터는 2대 혹은 3대이고, 프린터는 1대뿐이라 불편하게 사용하고 계십니까?
- 컴퓨터 1대에 프린터를 1대씩 사용하거나 추가 구입시 프린터를 구매하려고 생각하십니까?
- 프린터를 하고 있는 동안 컴퓨터를 장시간 사용하지 못하고 계십니까?
- CAD 사용시 Plotter가 그리고 있는 동안 컴퓨터를 장시간 사용하지 못하고 계십니까?
- 컴퓨터의 수량과 프린터의 수량이 안맞아 적절한 사용을 못하고 계십니까?
- 학교 및 학원의 실습실에 프린터대수가 부족해 적절한 교육을 못하고 계십니까?
- F.A 혹은 계측기쪽의 프린터 인터페이스에 문제가 있읍니까?

컴퓨터 사용중에 불편한 위의 모든 문제를 자신있게 처리해 드리므로, 귀사의 업무에 도움이 되어 드리고 싶읍니다!



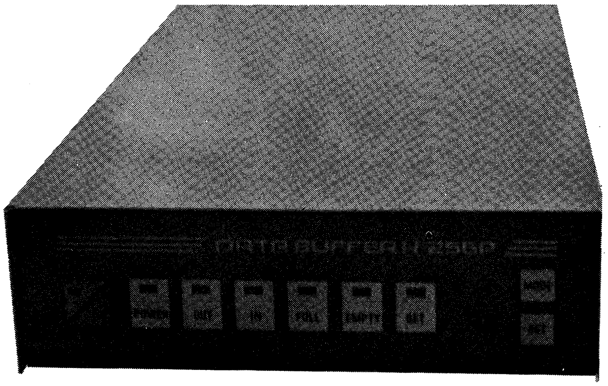
※ K-P-345A는 컴퓨터 프로그램 보호법에 의해 등록된 제품입니다.

K-P-345A의 사양

입·출력 방식	Centronics Type
입력 COMPUTER	3 : 1~5 : 1
출력 PRINTER	1 대
대기 시간	5초, 10초, 20초, 40초 중 선택
채널 변환방식	전자식
채널 선택방식	자동 or 반자동
용지 F.F 기능	자동 F.F or None F.F
F.F 삭제 기능	자동 삭제 or None

K-256P의 사양

버퍼 용량	256kB(80col 약 200장)
입·출력 방식	Centronics Type
입·출력 수	1 : 1 or 2 : 1
사용자 MODE	10EA
고유 기능	Data 중 Space 입력 기능, 재 Copy 기능, 메모리 기능, Self Test 기능 등 총 10가지



● DATA SWITCH
- 전자식 완전자동 (2 : 1~8 : 1 or 64 : 1)
- 전자식 반자동 (2 : 1~8 : 1, 4 : 2)
- Serial 완전자동(4 : 1, 8 : 1)

● DATA BUFFER
- 소용량 64kB
- 고성능 다기능 256kB
- 대용량 다기능 1MB
- Serial용 1MB

● DATA CONVERTOR
- Serial→PARALLEL
- PARALLEL→Serial

● 각급 학교 및 학원의 전산 실습실 프린터 공유장치 공사

ATTA SYSTEM
PRINTER 주변기기 전문메이커
서울·종로구 종로3가 175-4 (현대APT1309호)
전 화 : 2 7 7 - 1 7 3 9

● 상기 제품은 아래의 전문 취급점에서 전시 판매하고 있으며 직접 사용해 보실 수 있읍니다.●

서울 : 세운상가 협진 274—5419(가동 라열 426호)
용산 나진상가 동해전자 719—8010(17동 가열 304호)
용산 전자랜드 협진 712—2764(3층 C-336호)

인천 : 바이너 리컴퓨터 (032)762-5544
대구 : 범진 컴퓨터 (053)252-2211
마산 : 한길 컴퓨터 (0551)94-5555
원주 : 중부 시스템 (0371)43-7979

(기타 지역 전문 취급점 모집중)

새롭게 부각되는 매킨토시의

1984년 처음으로 일반에게 선을 보인 매킨토시는 발표당시 CPU에는 MC68000을 채용하였고 64k바이트의 ROM, 128k바이트의 RAM, 400k바이트 용량의 디스크를 탑재한 소규모의 시스템이었다.
그런데 이러한 매킨토시가 최근들어 새롭게 각광을 받고 있는 것은 하드웨어적인 우수성이라기 보다는 확장성을 고려한 다양한 유저 인터페이스 기능 때문이다.

매킨토시가 「윈도우+아이콘+마우스」라고 하는 유저·인터페이스를 퍼스컴에 정착시켰다고 하는 의의는 매우 크다.

매킨토시란?

매킨토시(Macintosh)의 사용자는 대부분 매킨토시를 맥(Mac)이라고 통칭하여 부른다. 원래 매킨토시라는 이름은 카나다산의 작은 청사과의 이름에서 따왔다고 한다. 사과의 이름을 제품명으로 할 정도로 애플사의 맥에 대한 기대는 상당한 것이였다.

매킨토시는 1984년 처음으로 일반에게 선을 보였다. 당시는 CPU에 MC68000, 64K바이트의 ROM, 128K바이트의 RAM, 400K바이트용량의 디스크를 탑재하고 있었다. 지금 생각하면 대단히 작은 구성이지만 맥의 정신은 이때 탄생하여 지금에도 계승되고 있다. 그후 버전업을 계속하여 RAM이 512K바이트인 Mac 512K가 발표되고 1986년에는 128K바이트의 ROM, 1M바이트의 RAM, 800K바이트의 디스크를 탑재한 Mac Plus가 발표되었다.

최근에는 처리속도를 약간 빠르게 하여 하드웨어의 확장성을 목표로 한 Mac SE와 CPU에 MC68020을 탑재하여 워크스테이션의 기능을 갖게 한 Mac Ⅱ 까지 등장하게 되었다.

Smalltalk의 영향

Smalltalk의 참신한 아이디어중 다음의 3가지를 맥에서 볼 수 있다.

① 비트맵·디스플레이와 마우스의 채용
② 일관성이 있는 유저·인터페이스
③ 시스템에서 제공하는 풍부한 루틴

①의 비트맵·디스플레이를 채용하므로써 종래의 컴퓨터에서 볼 수 있던 것 같은 문자베이스의 표시에서 그래픽스를 베이스로 한 표시가 가능하게 됨과 동시에 문자의 표시에도 폰트를 변경함으로써 가지각색의 글자체로 표시하는 것이 가능하게 되었다. 또한 마우스의 채용에 의해 사람에게 우수한 유저·인터페이스를 제공할 수 있도록 되었다. 마우스와 마찬가지인 포인팅·디바이스에는 트랙 볼이나 조이스틱, 라이트펜 등 몇가지가 있지만 이들중에서 마우스의 유용성을 실증한 공적은 매우 크다고 볼 수 있다.

②의 일관성이 있는 유저·인터페이스는 ①의 상승효과로 매우 기억하기 쉽고 조작성이 좋은 환경을 제공하고 있다. 이것에 의해 컴퓨터 엔지니어가 아닌 엔드 유저도 쾌적하게 맥을 사용할 수 있게 되었다.

Smalltalk에서는 팝업 메뉴(그림 1)로 구성되지만 맥에서는 풀다운 메뉴로 화면이 구성되어 있다. 전자는 커서가 있는 장소에 매뉴가 표시되는 방법, 후자는 화면의 상부에 있는 시스템 메뉴에서 서브메뉴를 표시하는 방법을 취하고 있다. 이 2가지는 메뉴를 표시하는 방법에는 차이가 있지만 본질은 같은 것이다.

③의 효용은 ②의 결과를 가져오는 복선으로도 작용한다. 맥에서는 ROM에 툴박스(Tool Box)라고 하는 이름의 루틴을 제공하고 있다(일부는 RAM이며 시스템·디스크로 제공되는 것도 있다.)

ToolBox에는 맥을 사용하기 위한 기본적인 처리가 모두 마련되어 있으므로 프로그래머는 자유로이 맥을 능숙하게 사용할 수 있음과 동시에 유저·인터페이스의 부분도 서포트하고 있으므로 누가 만들어도 마찬가지인 조작성을 유지할 수 있다. 이 외에 디지타이징한 음을 취급하는 사운드·프로세서를 내장하고 있다.

Smalltalk의 아이디어중 가장 유명한 것은 무엇보다도 오브젝트지향성이라고 할 수 있다.

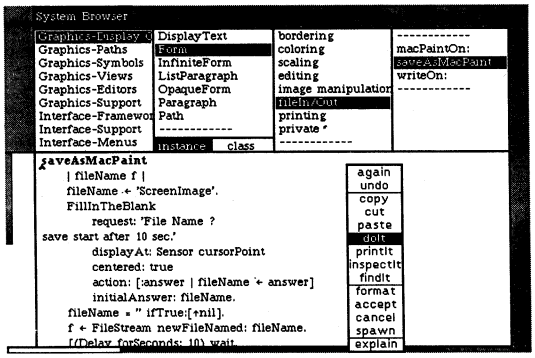
오브젝트지향이란?

오브젝트지향이라고 하는 말의 본질을 정확하게 이해하고 있는 사람은 그다지 흔치 않은 것 같다. 오브젝트지향이란 「어떤 물건을 중심으로 생각하고, 그 물건에 메시지를 보냄으로써 물건의 상태가 변화한다.」고 하는 단순한 사고방식이다(그림 2 참조). 이 개념을 프로그래밍언어 및 시스템의 세계에 가지고 들어와서 또한 효과적으로 이용하고 있는 것이 Smalltalk이다. 그러나 또 하나 놓쳐서는 안될 것은 유저 인터페이스의 세계에도 이 개념이 도입되어 있다고 하는 것이다.

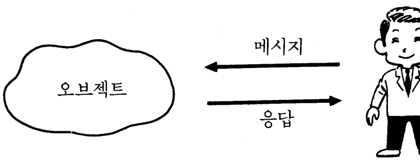
Smalltalk에서는 이들이 실로 잘 융합되어 있으므로 구별하여 생각하기는 실은 오브젝트지향

유저 인터페이스 기능

〔그림 1〕 팝업 메뉴



〔그림 2〕 오브젝트 지향성



이라는 것은 다음과 같이 경우에 따라서 나눌 수 있다.

① 오브젝트지향설계
② 오브젝트지향 프로그래밍
③ 오브젝트지향 유저인터페이스

①은 프로그램의 설계단계이다. 일반적으로 프로그램을 설계할 때에는 우선 데이터를 생각하고, 그 데이터를 「어떻게 처리하여 갈 것인가?」하는 구상을 세우는 경우가 일반적이라고 생각한다. 따라서 각각의 처리내용은 데이터마다 개별적으로 생각하여 종합된다. 이 자연적인 사고방식이 바로 오브젝트지향인 것이다.

②의 오브젝트지향 프로그래밍에서는 데이터마다 개별로 생각한 「처리내용」과 「데이터」를 하나의 「클래스」라고 하는 하나의 그릇에 국소적으로 관리할 수 있다. 또한 클래스에 계층구조를 가지게 함으로써 계승의 효과가 나타나며, 프로그래밍에 다양성이 생긴다. 그러므로 오브젝트지향 프로그래밍에서는 설계에서 프로그래밍까지 자연적인 흐름으로 실현할 수 있다.

③의 오브젝트지향 유저 인터페이스는 우선 「물건」을 선택하고 나서 그것을 「어떻게 한다」(메시지송신)고 하는 유저 인터페이스를 제공한다. 코맨드 베이스의 유저 인터페이스에서 「어떻게 한다」, 「무엇을」이라고 하는 경우와는 대조적이다.

이와 같이 한마디로 「오브젝트지향」이라고 하여도 이를 인식하는데 여러가지 방법이 있으므로 정확하게 인식하는 것이 매우 중요하다. 즉 ①, ②, ③의 성질은 서로 독립하여 존재하고 있다. ①은 사고방식의 문제이므로 모든 설계에 있어서 적용이 가능하다. Smalltalk에서는 ②, ③이 융합하여 제공되고 있다. 「클래스」라는 개념은 「오브젝트지향 프로그래밍」의 단계로 되어 「제조과정」으로서 등장하는 것이며, 오브젝트지향이라고 하는 것을 정의하는 요소는 아니다. 맥에서는 ③을 「시스템」이 제공하고 있다. 그러므로 매킨토시는 우수한 유저인터페이스를 가지고 있다라고 이야기할 수 있는 것이다. 왜냐하면 「오브젝트지향」은 인간에게 자연스러운 사고과정이기 때문이다.

기본조작

맥에서는 마우스를 조작함으로써 키입력 이외의 모든 처리를 지시할 수 있다(소프트 키보드를 사용하면 키입력도 마우스로 할 수 있지만 그런 사람은 없을 것이다.) 우선 기본적인 마우스조작부터 설명한다.

① 커서이동 : 버튼을 누르지 않고 마우스를 움직이게 한다.
② 클릭 : 버튼을 재빨리 눌렀다 뗀다.
③ 더블 클릭 : 버튼을 연속해서 재깍재깍 2회 클릭한다.
④ 드래그 : 버튼을 누르면서 마우스를 움직이게 한다.

이들 4가지의 조작으로 기본적인 것은 모두 할 수 있다. 높은 수준의 테크닉으로서 시프트키, 옵션키, 크로버키를 누르면서 마우스 조작을 할 경우도 있다. 맥을 처음 사용하는 사람은 더블 클릭의 조작이 상당히 낯설지만 익숙하면 매우 편리하다.

파인더

맥을 구동시켰을 때에 가장 눈에 걸리는 것은 파인더이다. 이 상태는 MS-DOS에서 시스템을 구동시켰을 때에 프롬프트가 나타나는 상태와 같다. 파인더에서는 프로그램의 기동이나 화일의 관리 등을 하며 볼륨이나 디렉토리, 화일 등이 아이콘으로 표시된다. 맥에서는 디렉토리에 상당하는 것을 「폴더」라고 하는데 볼륨이나 폴더의 내용은 윈도우를 사용해서 표시되며 그들의 윈도우는 자유로이 오버랩하여 멀티윈도우를 구성한다.

「오브젝트지향이란」에서 설명한 바와 같이 파인더에서 아이콘은 「물건」에 상당한다고 할 있다(그림 3 참조).

●화일정보의 표시

임의의 볼륨이나 폴더에 커서를 이동하여 더블·클릭 또는 화일메뉴에서 「연다」(open)를 지시하면 윈도우가 열려진다. 그 때에 표시되는 아이콘은 화일정보를 나타낸다.

●상세한 화일정보의 표시

「일람표」(view) 메뉴에서 「이름」(name), 「날짜」(date), 「크

기」(size), 「종류」(kind)를 지시하면 DIR/W와 마찬가지로 리스트표시를 각각의 항목에서 소트하여 얻을 수 있다.

● 화일의 카피

동일 폴더내에 복제를 만드는 경우는 아이콘을 선택하여 화일 메뉴에서 「복제」(Duplicate)를 지시한다. 볼륨간의 경우는 아이콘을 드래그하여 카피선에서 마우스 버튼을 뗀다. 이때 카피선의 아이콘이 볼륨 또는 폴더의 경우는 수납을 나타내는 의미에서 아이콘이 검게 하일라이트된다.

● 화일의 이동

이름의 변경을 하고 싶은 경우는 아이콘의 아래에 있는 해당 이름으로 커서를 이동하여 클릭하면 커서의 형상이 바뀌는데 이때 임의의 이름을 키인한

다른 폴더로 이동하고 싶을 때는 아이콘을 드래그하여 이동선에서 마우스 버튼을 뗀다. 이 조작은 볼륨간의 카피와 같다. 즉, 같은 조작을 볼륨간에서 하면 카피로 되고, 동일 볼륨의 폴더 간에서 하면 이동으로 된다.

● 화일의 소거

아이콘을 드래그하여 「쓰레기통」(Trush)에서 마우스 버튼을 뗀다. 이것은 볼륨간의 카피 작업을 「쓰레기통」이라고 하는 볼륨에 대해서 하는 것으로 된다. 한번 쓰레기통에 버린 것은 다음에 프로그램을 기동하기까지 혹은 명시적으로 특별메뉴의 「쓰레기통을 비게 한다」(Empty Trush)를 지시하기까지는 남아있어서 아이콘이 불룩한 상태로 되어 있다.

● 디렉토리의 생성

화일메뉴의 「신규폴더」를 지시함으로써 새로운 폴더가 생성된다.

● 프로그램의 기동

아이콘을 선택하여 더블 클릭 또는 화일에서 Open를 지시하면 프로그램이 기동된다. 이 오퍼레이션은 「화일정보의 표시(DIR)」와 같다. 다른 것은 볼륨이나 폴더이외의 아이콘인 것이다(**그림 4** 참조).

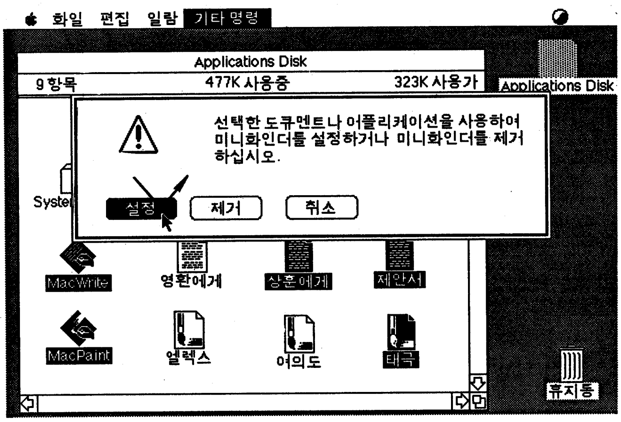
오퍼레이션 (동일의 메시지)이어도 선택한 아이콘(물체)에 의해서 그 행동은 다르게 된다. 즉 메시지를 받았을 때에 어떻게 행동하면 되는가를 「물체」는 알고 있다고 하는 오브젝트 지향의 정신이 여기에 활용되고 있다. 프로그램의 아이콘을 선택하면 데이터가 없는 상태에서 프로그램이 기동된다. 데이터의 아이콘을 선택하면 그 데이터를 읽어 들인 상태에서 프로그램이 기동된다. 이것은 MS-DOS의 코맨드 등에서 아규먼트를 지정하지 않고 프로그램을 기동하였을 경우와 아규먼트를 지정하여 프로그램을 기동하였을 경우에 해당한다.

맥에서는 데이터 화일이 「자신이 어떤 프로그램에 의해서 만들어졌는가?」고 하는 것을 데이터자신이 기억하고 있어서 그것을 아이콘형상으로 표시하고 있으므로 데이터의 취급이 매우 간단해진다. MS-DOS와 같이 코맨드와 데이터의 조합을 생각하면서 기동할 필요는 없다.

데스크 액세서리

데스크 액세서리(Desk Accessory : 약 DA)는 시스템 메뉴의 사과마크를 눌렀을 때에 나타나는 애플 메뉴에 등록되어 있는 편리한 프로그램이다. 맥은 원래 싱글 프로세스의 시스템이므로 원칙적으로 하나의 프로그램을 종료하지 않고 다른 프로그램을

〔그림 3〕 파인더의 설정



기동할 수는 없다. DA는 이 제약을 받지 않고 언제라도 기동할 수 있다. 맥시스템을 셋업하기 위한 콘트롤 판넬 (Control Panel)이나 프린터를 선택하는 셀렉터는 맥을 사용하는데에 없어서는 안되는 것이다.

스크랩 북

스크랩 북은 프로그램사이에서 데이터를 이동시키는데에 사용한다. MacDraw에서는 그리지 않는 비트맵의 그림을 MacPaint로 그려서 그것을 MacDraw 속에 거두어들이는 곡예를 할 수 있다.

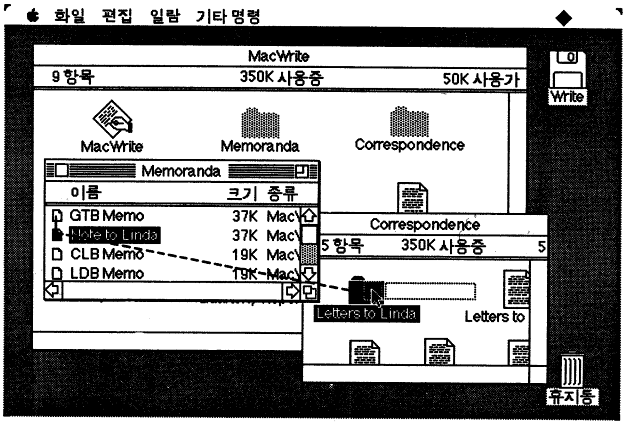
맥에서는 DTP(Desk Top Publishing)의 툴이 수없이 많이 있다. DTP의 툴은 레이아웃이 가능하며 그 툴자체에서는 그다지 그림그리기 기능을 가지고 있지 않다. 맥에서는 가질 필요가 없는 것이다. 다른 시스템에서 DTP툴자신이 작화를 시작하여 모든 기능을 함께 담은 올인원방식과는 대조적이다.

각 툴에서 작성한 그림을 모두 스크랩 북에 넣어 두면 DTP툴에서 거두어들일 수 있다.

맥에 있어서 DTP가 보급한 큰 요인으로서 이 스크랩 북의 기능을 놓칠 수는 없다. 그래픽스를 가장 빨리 시스템에 채용하고 또한 스크랩 북이라고 하는 구조를 마련한 것은 맥만의 장점이다. 실은 이 구조를 뒤에서 지탱하고 있는 것은 앞에서 설명한 ToolBox인 것이다. 맥의 화면에 그림을 표시하려면 모두 ToolBox 속의 QuickDraw에서 사용하는 PICT데이터의 형식으로 할 필요가 있다. 스크랩 북은 이 PICT데이터를 유지하고 있다.

맥에 있어서의 거의 모든 프로그램은 표시하고 있는 그림에 대한 컷 & 페이스트기능을 가지고 있다. 편집메뉴의 컷 또는 카피를 하면 대상으로 하는 그림의 정보는 클립보드 화일(Clip Board File)에 대피된다. 페이스트기능은 클립보드의 정보를 사용하고 있다. 한편 스크랩 북도 마찬가지로 이 클립보드 화일을 경유하여 데이터의 컷과 페이스트를 하고 있다. 즉 컷과 페이스트 기능을 가진 프로그램이면 스크랩 북을 의식하지 않고 스크랩 북을 이용할 수 있는 것이다. 이와 같은 구조는 다른

〔그림 4〕 아이콘의 사용



데서는 볼 수 없다.

매니저

맥의 프로그램을 이야기할 때에 자주 「---매니저」라는 말을 듣는다. 대표적인 매니저에는 메모리매니저, 윈도우매니저, 콘트롤매니저, 메뉴매니저, 다이얼로그매니저, 리소스매니저 등이 있다. 이들은 ToolBox내부를 기능별로 분류한 호칭법이다.

ToolBox는 128K바이트의 ROM의 속에 들어 있는 하나의 라이브러리이다.

예를 들면 「윈도우 매니저」라고 하면 「ToolBox 속의 윈도우를 조작하기 위한 루틴군」을 의미한다.

앞으로의 동향

MacⅡ는 덩치는 크고, 여러가지 다양한 기능을 가지고 있으며 지금까지의 Mac Plus나 Mac SE와는 조금 다른 개념을 가진 시스템이다. 그도 그럴 것이 MacⅡ는 애플사가 워크스테이션분야에 참여하기 위한 첫번째 작품으로 만든 것이기 때문이다. Mac Plus와 Mac SE는 IBM PC 등의 퍼스널컴퓨터 분야에, Mac Ⅱ는 SUN 워크스테이션이나 NeXT 등의 워크스테이션 부류에 속한다고 할 수 있다. 1989년 중반기 중으로는 랩탑분야에도 참여하기 위해서 랩탑형의 맥도 발표될 예정이다. 이것은 인터페이스가 종래의 마우스에서 트랙볼로 변경된다(역시 무릎의 위에 얹는다면 마우스는 사용하기 어려울 것이다). 시기는 늦어도 1989년 9월중에는 발표한다고 애플사에서는 발표하고 있다.

●Mac Plus / Mac SE

우선 퍼스널컴퓨터로 분류되는 MacPlus는 앞으로 버전업은 없으리라 생각된다. SE의 출현으로 그 역할은 끝났다고 볼 수 있다. 또 Plus는 신규생산되고 있지 않다. SE의 출현은 자동차의 세계에서 말하면 풀 체인지와 같은 것이며, 지금까지 Plus가 담당하고 있던 분야를 대신하는 의미로 해석된다. 전략적으로 본 경우 SE는 시스템의 확장성을 높이는 연구가 계속되고 있으며 동시에 가격의 다운도 꾀해지고 있다.

SE는 앞으로 2M바이트의 RAM이 표준으로 장착되고 40M바이트의 하드디스크(HD)도 가까운 시일안에 내장되리라 생각된다. 또한 플로피디스크 드라이브를 SuperDrive(나중에 설명)로 채용한 것도 발표되었다. 더우기 SE의 선배로서 68030 SE도 발표되어 지금의 Mac Ⅱ가 보여주고 있는 스피드이상의 것을 제공할 예정으로 되어 있다.

●Mac Ⅱ

워크스테이션 분야에서는 굉장한 변화가 있다. 지난 88년 9월에는 CPU로 68030을 채용하였고 SuperDrive를 탑재한 Mac Ⅱ의 선배인 Mac Ⅱx도 발표되었다. 지금까지 맥에서는 3.5인치 2DD의 FDD를사용하고 있으며, 이것은 0.8M바이트밖에 들어 있지 않다. SuperDrive는 3.5인치 2HD의 FDD도 사용할 수 있도록 되어 1.4M바이트까지 들어간다. 또한 SuperDrive는 MS-DOS포맷의 디스크도 드라이브할 수 있고, MS-DOS에서 만든 데이터를 직접 읽을 수도 있다.

Mac Ⅱx는 68030으로 되어 있지만 동작클럭이 Mac Ⅱ의 68020과 같은 16MHz이므로 처리속도는 1할에서 2할정도 밖에 빨라지지 않을 것 같다(68030은 본래는 20MHz로 동작한다). CPU의 처리능력면에서 볼 때 2배정도 빨라진 것은 어떻게 이루어질 것인가? 움직이지 않는 소프트웨어가 많아져 버리는 것인지, 그렇지 않으면 전략적인 것인지 상세하게는 알 수 없다.

Mac Ⅱx를 발표하는 의미는 스피드업은 아니고 「68030머신을 애플사가 발표하였다」고 하는 것에 있는 것 같다. 최근 NeXT나 각사에서 68030머신을 발표하고 있고, 애플로서도 가까운 장래 68030머신이 주축으로 되어 갈 것이 예상된다. 따라서 출발이 늦지 않다는 것을 어필하기 위해서 발표한 것으로 보는 측도 있고, 또 코스트삭감을 겨냥하여 68030머신을 발표하였다고 보는 측도 있다. 즉 장래 멀티태스킹 등을 하는 것을 생각하면 68020에서는 MMU(-Memory Management Unit)가 필요하게 된다. 68030에서는 자신이 MMU의 기능을 가지고 있으므로 68020에 MMU를 부가하기보다는 코스트적으로 싼 값으로 할 수 있다는 말이 된다.

●Mac Ⅱ / Ⅱx 의 후계기종

현재로는 여러가지로 강력한 기능을 가지고 있는 Mac Ⅱ와 Mac Ⅱx이지만 실은 이 2기종은 1989년중에 생산중지될 예정이다. 이미 다음 기종이 결정되어 있어서 Mac Ⅱ는 모델명이 「Stingray」로 Mac Ⅱx는 모델명 「Four Square」로 바뀔 예정이다.

Stingray는 지금의 Mac Ⅱ보다도 보드를 작게 하며 CPU에 68030, 연산프로세서에 68882, 3개의 NuBus슬롯이라고 하는 구성으로 되어 가격은 Mac Ⅱ보다도 저렴하게 될 예정이다. 공급시기는 1989년 4월 이후가 될 것이다.

Four Square는 지금의 Mac Ⅱx와 같은 보드이며 CPU에 20MHz 의 68030에 커스텀 I/O칩이 붙을 예정이다. 이 커스텀I/O칩이 어떠한 것인가는 명확하게 되어 있지 않다. 또 멀티태스킹을 서포트할 예정이다. 시기는 1989년 9월 이후이다.

또한 1989년 10월 이후에 대용량의 디스크와 시큐리티를 확장한 데스크 사이드형의 모델명이 「Tower」인 저가격의 컬러판 Mac과 모델명이 「K-12」인 기종이 발표될 예정으로 되어 있다.

QUICKEY.COM의 분석

김대순/아주대 A-MICOM

QUICKEY.COM은 얼마전 국내의 모 컴퓨터 잡지에 소개된 적이 있는 프로그램이다. 이것은 길이가 불과 273바이트 밖에 되지 않는 램상주 프로그램으로서 커서를 자유롭게 사용할 수 있게 해주는 아주 깜찍한 프로그램이다. 따라서 여기서는 이러한 램상주 프로그램이 어떠한 원리로 작성되어 있으며, 어떻게 프로그램이 짜여져 있는지 상세하게 분석해 보았다.

대부분의 독자들은 컴퓨터앞에 앉아있는 시간의 대부분을 에디터를 사용하는데 할애하고 있을 것이다. 필자도 에디터를 사용하여 여러가지 편집작업을 하는데 필요에 따라 Arrow. Key를 반복해서 조작하곤 한다.

그러나 원하는 곳으로 커서를 움직이는데 있어 그 커서의 움직임이 얼마나 둔하지 필자와 같이 성격이 급한 독자들은 짜증을 느꼈을 것이다.

그런데 얼마전 QUICKEY.COM(마이크로 소프트웨어 86년 9월호에 의해 국내에 소개되었던 램상주 유틸리티로서 273 바이트의 아주 작은 코드로 키보드의 감도를 빠르게 해준다.)이라는 아주 깜찍한 프로그램이 커서를 날아다니게끔 해주면서 이러한 짜증을 시원해 해결해 주었다. 또한, 어떠한 조건(?)에서도 System Crash를 야기하지 않는 뛰어난 안정성을 가지고 있으며, 여러번 실행할수록 키보드의 감도가 더욱 빨라져 필자를 놀라게 하였다,.

결국 이러한 원인이 필자로 하여금 "이 유틸리티가 어떻게 키보드의 감도를 빠르게 해주는가"하는 의문을 가지게 하였다. 따라서 그 알고리즘을 디버거를 이용하여 분석해 보았다(아마 관심있는 독자들은 이미 분석해 보았으리라 믿는다).

독자들도 디버거를 이용하여 소스리스트가 공개되지 않은 많은 유틸리티들을 분석해보려고 했을 것이다. 필자도 마음에 드는 간단한 프로그램이 있으면 그 프로그램의 알고리즘과 테크닉을 얻기 위해 이를 분석해보곤 한다. 이때 무엇보다도 제일 먼저 해야 할 작업은 디버거를 통해 데이터영역을 찾아내는 일이다(일반적으로 데이터영역은 프로그램의 앞부분이나 뒷부분에 존재한다). 그리고 이를 디스어셈블하여 어셈블리어로 표현된 기계어 리스트를 얻어 RET나 IRET로 각각의 프로시듀어를 구분한 다음 각 프로시듀어를 한 라인씩 실행하면서 분석한다.

또한 이때 중요한 것은 각 프로시듀어들 간의 관계를 CALL, JMP, JE, JNZ 등의 명령어와 각 프로시듀어들이 공통적으로 사용하는 변수를 통해 규명하는 것이다.

여기 소개하는 QUICKEY.ASM도 이와 같은 방법에 따라 완성할 수 있었다. QUICKEY.ASM은 타이머 인터럽트와 키보드 인터럽트를 가로챈 다음, 새로 정의한 인터럽트를 상주시키고, ROM BIOS DATA AREA를 조작함으로써 커서가 날으는 듯한 신속한 기능을 발휘한다. 간단히 얘기해서 같은 키가 연속적으로 눌려지면, 1초에 18.2번 발생하는 타이머 인터럽트가 그 키를 누른 것처럼 조작하는 것이다.

그러면 QUICKEY.ASM을 이해하기 위해 키보드 인터럽트, 그리고 ROM BIOS DATA중 키버퍼와 키보드 플래그에 대해 살펴보자. 참고로 프로그램을 이해하는데 필요한 8086 어셈블리와 IBM PC의 메모리구성 및 프로그램구성과 램상주 기법에 대해서는 퍼스널컴퓨터 89년 2월호 '어셈블리 강좌 (1)의 어셈블리 기본 개념'과 동지 '시스템 분석 코너의 IBM PC의 이해(2)-시스템 자원 (2)'에 자세한 설명이 되어 있으므로 이를 참고하면 이해가 빠를 것이다.

ROM BIOS 데이터 영역의 이용

'IBM PC Technical Referance'의 부록 A를 보면 ROM BIOS 리스트가 나와 있는데 라인 76부터 245라인까지 보면 ROM BIOS DATA AREA라는 데이터 세그먼트가 있다. 여기에는 키보드, 비디오 디스플레이, 디스켓 데이터 등 시스템 프로그래밍에 필요한 모든 데이터가 실려 있다. 실제로 우리는 이 데이터 세그먼트를 공유하여, 이 영역을 변수로 사용할 수 있다. 또 이것을 이용한 프로그램도 실제로 많이 볼 수 있다(리스트 1 참조).

필자가 지금 소개하고자하는 QUICKEY.ASM도 이 ROM BIOS 데이터 영역을 이용한 것이다. 그럼, 먼저 QUICKEY.

ASM이 이용하고 있는 키보드 플래그에 대해서 알아보기로 하자.

키보드 플래그

이 키보드 플래그는 메모리 0040 : 0017H(절대번지 00417H)에 존재하는 1바이트 데이터를 키보드 플래그(KB_FLAG)로 하여금 토글이나 시프트키 스트로크가 발생하면, 키보드 인터럽트에 의해 그 키의 상태를 Bit Mask하여 표시하도록 한다(그림 1 참조).

키버퍼

키버퍼 번지는 0040 : 001AH(절대번지 0041AH)번지 이후의 36 바이트를 키버퍼라 한다. 이 키버퍼의 첫번째 2바이트(절대번지 0041AH)는 출력할 데이터의 주소가 저장되고, 두번째 2바이트(절대번지 0041CH)는 데이터가 들어올 주소를 저장한다. 또한, 절대번지 0041AH 이후 4바이트에 저장되는 주소는 오프셋 어드레스를 가르킨다. 즉, 절대번지 0041CH 번지에 34H가 있다면, 입력하는 키의 아스키코드와 스캔 코드가 각각 절대번지 00434H, 00435H에 저장되는 것을 뜻한다(그림 2, 3 참조).

키버퍼에 저장되는 키 데이터는 Circular Queue(First In First Out Structure)의 구조로 되어 있다. 임의의 키가 눌려져 데이터가 저장되는 방식을 보면, 0040 : 001CH번지의 값은 2씩 증가하고, 키를 누르기전에 0040 : 001CH번지의 값이 가리키는 번지의 눌려진 키의 아스키코드와 스캔코드가 저장된다.

즉, 키를 누를 때마다 키버퍼에는 키 데이터가 수록되고, 데이터가 들어오는 주소는 2씩 증가하게 된다. 이때 증가한 데이터가 들어올 주소가 0040 : 003EH일때는 그 주소가 0040 : 001EH로 세트된다. 그리고 데이터를 출력할 때는 0040 : 001AH 에 저장되어 있는 어드레스를 2씩 증가시키며, 0040 : 001CH에 저장되어 있는 어드레스의 바로전까지 입력된 순서대로 출력한다. 그러므로 0040 : 001AH(Head Pointer), 0040 : 001CH(Tail Pointer)를 같게 하면, 키 입력이 없는 것을 뜻하고, 다르면 키 입력이 있는 것을 의미한다.

키보드 인터럽트

키보드로부터 키가 하나 눌려져 시스템 보드에 전달되는 과정을 보면, 키보드의 8048 마이크로프로세서가 키 스트로크를 인식하여 시스템보드에 있는 8255 칩포트 A의 키 스트로크에 스캔코드를 넘겨주고, 8255 칩이 "ACK"를 다시 8048에 보내준다. 이때 키보드

〔리스트 1〕 ROM BIOS 데이터영역의 일부

```
LOC   OBJ    LINE   SOURCE
----         80     40H에 데이터 세그먼트
                        .
                        .
0017  ??     93     KB_FLAG     DB    ?
                        .
                        .
0008         101    ALT_SHIF    EQU   08H
0002         103    LEFT_SHEFT  EQU   02H
                        .
                        .
0019  ????   115    BUFFER_HEAD DW    ?
001A  ????   116    BUFFER_TAIL DW    ?
001C  ????   117    KB_BUFFER   DW    ?
001E  (16               .
      ???               .
      )                 .
----         215    DATA ENDS
```

〔그림 1〕 키보드 플래그(0040 : 0017h)

비트	키	비트가 1 인 경우
7	Insert	Insert 모드 on
6	Caps Lock	Caps Lock 모드 on
5	Num Lock	Num Lock 모드 on
4	Scroll Lock	Scroll Lock 모드 on
3	Alt Shift	key down
2	Ctrl Shift	key down
1	Lefthand Shift	key down
0	Righthand Shift	key down

〔그림 2〕 키버퍼의 덤프

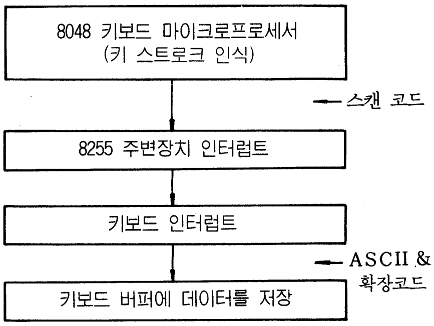
```
-d 0040:0017
0040:0010                         00-00 00 32 00 32 00 30 0B        ...2.2.0.
0040:0020  30 0B 34 05 30 0B 3A 27-30 0B 30 0B 31 02 37 08  0.4.0.:'0.0.1.7.
0040:0030  0D 1C 64 20 6F 18 6F 18-08 0E 08 0E 20 39 00 00  ..d 0.0..... 9..
0040:0040  8D 00 C0 03 04 DC 01 00-00 07 50 00 00 40 00 00  .........P..@..
```

〔그림 3〕 키버퍼에 데이터 'BUFFER'가 수록되어 있는 경우

1A	1C	1E	20	22	24	26	28	2A	2C	2E	30	32	34	36	38	3A	3C
34	28							R	E	F	F	U	B				

(1C) TAIL POINTER
(1A) HEAD POINTER

〔그림 4〕 키보드 인터럽트 작용의 흐름



인터럽트(INT 9H)가 발생하는 것이다(**그림 4** 참조).

사실 IBM PC에서는 2개의 키보드 인터럽트(INT 9H, INT 16H)를 사용하고 있는데, INT 9H는 위에서 설명한 과정에 의해 포트 60H에 저장되는 스캔코드를 캐릭터코드로 변환시켜 키 버퍼에 놓는 일을 수행한다(ROM BIOS 리스트 1843부터 2273 라인까지의 KB_INT 프로시듀어). 또한, 〈Ctrl-Alt-Del〉에 의한 시스템 리셋과 〈Shift-Prtsc〉에 의한 하드카피 등도 수행한다.

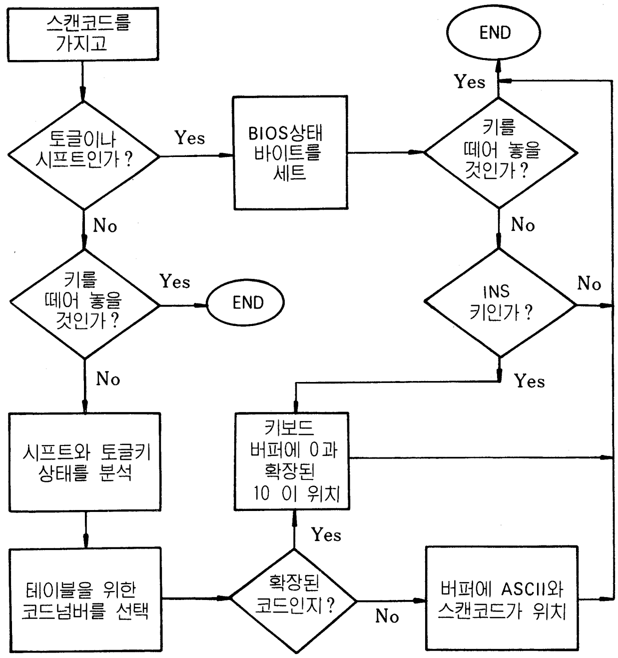
INT 16H는 KB_INT(INT 9H)에 의해 채워지는 키버퍼를 검사하여 키 입력을 기다린 다음 INT 16H(BIOS 리스트 1195-2006, 2146-2153 라인의 KEYBOARD_IO Procedure)를 이용하는 루틴에 새로운 키의 입력을 전달하는 역할을 맡는다. 여기서 우리가 QUICKEY.ASM을 이해하는데 필요한 것은 INT 9H 루틴이므로, INT 16H에 대한 더이상의 자세한 설명은 피하기로 한다(관심있는 독자는 ROM BIOS 리스트를 보기 바란다).

KB_INT Procedure(INT 9H)는 포트 60H에서 스캔코드를 읽어 먼저 그 키 스트로크가 시프트나 토글키인지를 판단한다. 만약 시프트나 토글키인 경우 BIOS STATUS BYTES를 세트하고 그 눌린 키가 풀렸는지 판단하여 풀렸을 경우 KB_INT를 끝내고 그렇지 않을 경우 키가 〈INS〉인가를 비교하여 2차 코드로 키버퍼에 코드를 저장하는 것을 결정하게 된다.

그리고 위에서 시프트나 토글키가 아닐 경우 그 키가 풀렸으면 KB_INT를 중단하고 풀리지 않았을 경우 시프트나 토글키 상태를 검사하여 코드 테이블로부터 코드를 찾아 그것이 2차 코드인지를 판단하여 아스키코드와 스캔코드를 버퍼에 저장할 것인가 아니면 0과 2차 코드를 버퍼에 저장할 것인가를 결정한다(**그림 5** 참조).

위에 정리한 키보드 플래그와 키보드 버퍼 그리고 키보드 인터럽트만 이해한다면, QUICKEY.ASM을 이해하는데는 별 무리가 없을 것이다. 독자의 이해를 돕기 위해 QUICKEY.ASM 소스리스트에 주석을 달아 뒤에 실어 놓았다.

〔그림 5〕 키보드 인터럽트의 흐름도



QUICKEY.ASM의 구조

●인스톨 루틴

NewKbd와 NewTimer가 각각 원래의 인터럽트 9, 8번지를 갖게 하고 각각의 루틴을 램에 상주시킨다.

●새로운 키보드의 인터럽트루틴

인스톨루틴(Install Routine)에 의해 원래의 인터럽트 9번지를 가지므로 아무 키나 누르면 이 루틴이 실행된다.

① 처리가능한 키인가를 나타내는 UsablekeyFlag를 초기화하고, 눌린 키의 코드가 들어올 곳의 어드레스를 대피시킨 후 원래의 키보드 인터럽트를 호출한다.

② 오버런 코드이면 눌린 키를 취소하고 이 루틴을 빠져나온다.

③ 현재 눌린 키와 바로 전에 눌린 키를 비교하여 SameOther Flag와 Usablekey Flag를 세트하고, 현재 눌린 키를 버퍼에 저장한 후 NewKbd를 끝낸다.

●새로운 인터럽트루틴 타이머

NewKbd와 같은 양식으로 처리되었으므로 1초에 18.2번 이 루틴을 실행한다.

① NewKbd에 의해 세트된 플래그들을 검사한 후 새롭게 추가된 루틴을 실행할 것인지 판단한다.

② 눌린 키가 들어올 곳의 어드레스를 대피시킨 후 버퍼 포인터를 증가시킨다.

③ 처리해야 할 키가 존재하지 않으면 UsableKeyFlag를 '0'으로 한 후 원래의 인터럽트 타이머를 실행한다.

④ NewKbd에 의해 버퍼에 저장된 키를 입력한 후, SameOther Flag를 같게 한다.

끝맺으면서

QUICKEY.COM에는 〈Alt〉키와 함께 키보드 우측에 있는 뉴메릭 키로 아스키코드를 친 후 〈Alt〉키를 다시 떼었을 때 그 문자가 다시 〈Alt〉나 〈Shift〉키가 눌릴 때까지 반복하여 그 코드가 찍히는 버그가 있는데, 그 버그의 수정방법은 이미 공개되어 있으므로 여기서는 생략하기로 하고, 그 원인에 대해서 알아보자.

〈Alt〉키와 뉴메릭 키가 함께 사용되고 있는 중에는, 즉 〈Alt〉키로서 유저가 입력하고자 하는 코드번호를 입력하고 있는 중에는 헤드 포인터와 테일 포인터가 서로 같아 키 입력이 없는 것이 되지만, 누르고 있던 〈Alt〉키를 놓게되면 그 포인터들이 서로 다른 곳을 가르키게 되어 아직 처리하지 않는 키가 키버퍼에 존재하게 된다.

이때, QUICKEY .COM의 새로 정의한 키보드 인터럽트는 사용할 수 있는 키플래그를 1로 세트하게 되며, 타이머 인터럽트는 키버퍼 헤드와 키버퍼 테일의 값이 서로 달라 키를 키버퍼에 저장하는 부분을 실행하게 되어 〈Alt〉키와 함께 눌린 코드가 타이머 인터럽트에 따라 반복되어 쓰여지는 것이다.

이미, 국내에 소개된지 오래되어 QUICKEY.COM을 보유하고 있는 독자들도 많겠으나, 아직 갖고 있지 않은 독자들이나 이제 막 IBM-PC를 배우기 시작하는 독자들은 소스리스트를 한 라인씩 쳐가면서 QUICKEY.COM을 이해하기 바라며, 다음에는 더 좋은 글로 독자들과 만날 것을 약속한다.

〔리스트 2〕 QUICKEY.COM 소스 리스트

```
0000                    code    segment
                                assume cs:code

0100                            org 100h                        ; 100h 번지부터 시작

                        ;------------------ R O M  B I O S  D A T A  A R E A ----------------------

= 0040                        DataSegment      equ    0040h     ; Bios Data 영역 지정
= 0017                        KbdFlag          equ    0017h     ; 키보드 플래그 지정
= 0008                        AltShift         equ    0008h     ; <Alt> 키가 눌림
= 0002                        LeftShift        equ    0002h     ; 왼쪽 <shift>키가 눌림
= 001A                        KeyBufferHead    equ    001Ah     ; 키버퍼 데이터의 시작을 지목
= 001C                        KeyBufferTail    equ    001Ch     ; 키버퍼 데이터의 끝을 지목
= 001E                        KeyBufferStart   equ    001Eh     ; 키버퍼의 시작번지
= 003E                        KeyBufferEnd     equ    003Eh     ; 키버퍼의 끝번지

                        ;---------------- S t a r t  P o s i t i o n  H e r e ! --------------------

0100  E9 01C9 R         start:  jmp    install                  ; 설치 루틴으로

0103  ????????                  Old8Int         dd    ?         ; 8 번 인터럽트 벡터 저장 장소
0107  ????????                  Old9Int         dd    ?         ; 9 번 인터럽트 벡터 저장 장소
010B  ????                      Buffer          dw    ?         ; 입력된 키 저장 장소
010D  00                        UsableKeyFlag   db    00h       ; 처리 헤야 할 키인가 ?
010E  00                        SameOtherFlag   db    00h       ; 같은 키가 연속적으로 눌렸는가 ?
010F  07                        OtherKey        db    07h       ; 다른 키 구분자
0110  01                        SameKey         db    01h       ; 같은 키 구분자

                        ;-----------------------------------------------------------------------
                        ;               새 로 운   키 보 드   인 터 럽 트   루 틴
                        ;-----------------------------------------------------------------------
0111                    NewKbd    proc    far
```

```
                        ; 초 기 화 -- 새 키보드 인터럽트 루틴 초기화

0111  1E                      push    ds                              ; DS 저장
0112  53                      push    bx                              ; BX 저장
0113  BB 0040                 mov     bx, DataSegment                 ; Bios Data 영역 지정
0116  8E DB                   mov     ds, bx
0118  2E: C6 06 010D R 00     mov     UsableKeyFlag, 0                ; UsableKeyFlag  0

                        ; 눌린 키를 처리해야 할 것인가를 판단한 다음 플 래그들을 정리한다.

011E  8B 1E 001C              mov     bx, ds:KeyBufferTail            ; 키가 들어올 곳의 번지를 저장
0122  9C                      pushf                                   ; 플래그 레지스터 저장
0123  2E: FF 1E 0107 R        call    Old9Int                         ; 원래의 키보드 인터럽트를 부른다
0128  3B 1E 001C              cmp     bx, ds:KeyBufferTail            ; 키버퍼에 키가 입력 되었는가?
012C  75 1C                   jne     UsableKey                       ; yes, 키 처리루틴으로
012E  2E: C7 06 010B R FFFF   mov     Buffer, 0ffffh                  ; no, 버퍼에 Overrun 키를 저장

0135  F6 06 0017 02           test    ds:byte ptr KbdFlag, LeftShift  ; 왼쪽 <Shift>키가 눌렸는가 ?
013A  74 35                   je      NewKbdEnd                       ; yes, 키보드 인터럽트 취소
013C  F6 06 0017 08           test    ds:byte ptr KbdFlag, AltShift   ; <Alt>키가 눌렸는가 ?
0141  74 2E                   je      NewKbdEnd                       ; yes,  키보드 인터럽트 취소
0143  89 1E 001A              mov     ds:KeyBufferHead, bx            ; no, 키 입력이 없었던 것으로 한다.
0147  EB 28 90                jmp     NewKbdEnd                       ;     키보드 인터럽트 취소

                        ; 전에 눌린 키와 현재 눌린 키를 비교하여 그 결과 를 플래그에 정리한다.

014A                    UsableKey:
014A  8B 1F                   mov     bx, [bx]                        ; 현재 눌린 키의 데이터를 BX로
014C  0A FF                   or      bh, bh                          ; ASCII 코드를 가지고 있는가 ?
014E  74 21                   jz      NewKbdEnd                       ; no,  키보드 인터럽트 취소
0150  2E: 3B 1E 010B R        cmp     bx, Buffer                      ; yes, 전에 눌린 키와 같은가 ?
0155  2E: 89 1E 010B R        mov     Buffer, bx                      ; 현재 눌린 키를 저장
015A  2E: 8A 1E 010F R        mov     bl, OtherKey                    ; 다른 키가 눌린 것으로 한다
015F  75 05                   jne     KeyOther                        ; no, 다른 키 처리 루틴으로
0161  2E: 8A 1E 0110 R        mov     bl, SameKey                     ; yes, 같은 키가 눌린 것으로 한다
0166                    KeyOther:
0166  2E: 88 1E 010E R        mov     SameOtherFlag, bl               ; SameOtherFlag를 세트한다
016B  2E: C6 06 010D R 01     mov     UsableKeyFlag, 1                ; 처리해야 할 키로 한다.
0171                    NewKbdEnd:
0171  5B                      pop     bx                              ; BX 복구
0172  1F                      pop     ds                              ; DS 복구
0173  CF                      iret                                    ; 키보드 인터럽트를 끝낸다

0174                    NewKbd     endp

                        ;---------------------------------------------------------------------
                        ;               새 로 운  타 이 머  인 터 럽 트  루 틴
                        ;---------------------------------------------------------------------

0174                    NewTimer    proc    far

                        ; 새로 정의한 루틴을 실행할 것인지를 각 플래그들을 통해서 판단한다
```

```
0174  2E: 80 3E 010D R 01          cmp    UsableKeyFlag, 1          ; 처리 해야 할 키가 들어왔는가?
017A  75 48                        jne    OldTimer                  ; no, 원래의 타이머로
017C  2E: FE 0E 010E R             dec    SameOtherFlag             ; 다른 키가 들어왔는가 ?
0181  75 41                        jnz    OldTimer                  ; yes, 원래의 타이머로

                            ; 초 기 화 -- 새로운 타이머 인터럽트를 초기화한다

0183  1E                           push   ds                        ; DS 저장
0184  57                           push   di                        ; DI 저장
0185  53                           push   bx                        ; BX 저장
0186  50                           push   ax                        ; AX 저장
0187  BB 0040                      mov    bx, DataSegment           ; Bios Data 영역 지정
018A  8E DB                        mov    ds, bx

                            ;
                            ; 키버퍼 포인터를 증가 시킨다. ( tail pointer는 키가 들어올 곳의 주소를 의미한다. )
                            ;
018C  8B 1E 001C                   mov    bx, ds:KeyBufferTail      ; tail pointer를 BX로
0190  8B FB                        mov    di, bx                    ; 키가  들어올 곳의 번지를 저장
0192  83 C3 02                     add    bx, 0002h                 ; tail  pointer를 증가 시킨다.
0195  83 FB 3E                     cmp    bx, KeyBufferEnd          ; 증가한 tail pointer가 버퍼의 끝인가?
0198  75 03                        jne    Circular
019A  BB 001E                      mov    bx, KeyBufferStart      ; yes, tail pointer를 버퍼의 처음으로 한다
019D                        Circular:

                            ; 출력 해야 할 키가 존재하는가 ?

019D  3B 1E 001A                   cmp    bx, ds:KeyBufferHead      ; 출력할 키가 있는가 ?
01A1  75 09                        jne    KeyInput                  ; yes, KeyInput루틴으로
01A3  2E: C6 06 010D R 00          mov    UsableKeyFlag, 0          ; no, UsableKeyFlag를 0으로 한다.
01A9  EB 15 90                     jmp    NewTimerEnd               ; 타이머를 인터럽트를 취소한다.

                            ; 키보드 인터럽트에의해 들어온 키를 키버퍼에 저장한다.

01AC                        KeyInput:
01AC  FA                           cli                              ; 인터럽트를 불가능하게 한다.
01AD  2E: A1 010B R                mov    ax, Buffer                ; 버퍼에 놓인 키 값을
01B1  89 05                        mov    [di], ax                  ; 키버퍼에 입력한다.
01B3  89 1E 001C                   mov    ds:KeyBufferTail, bx      ; 키가 들어올 주소를 다음 번지로
01B7  FB                           sti                              ; 인터럽트를 가능하게 한다.
01B8  2E: A0 0110 R                mov    al, SameKey               ;
01BC  2E: A2 010E R                mov    SameOtherFlag, al         ; 같은 키가 눌린 것으로 한다.
01C0                        NewTimerEnd:
01C0  58                           pop    ax                        ; AX 복구
01C1  5B                           pop    bx                        ; BX 복구
01C2  5F                           pop    di                        ; DI 복구
01C3  1F                           pop    ds                        ; DS 복구
01C4                        OldTimer:
01C4  2E: FF 2E 0103 R             jmp    Old8Int                   ; 원래의 타이머로 간다.

01C9                        NewTimer    endp
```

```
                                  ;---------------------------------------------------------------------------
                                  ;                     Quick Key 설 치  루 틴
                                  ;---------------------------------------------------------------------------
01C9                              install  proc   near
                                           assume  ds:code

                                  ; 인터럽트 벡터 테이블을 지목하고, 원래의 인터럽트 벡터들을  대피시켜 놓는다.

01C9  33 C0                                xor    ax, ax                       ; AX를 0 으로 한다.
01CB  8E C0                                mov    es, ax                       ; 인터럽트 벡터 테이블을 지목
01CD  26: A1 0020                          mov    ax, es:[0020h]               ; 8번 인터럽트 벡터 오프셋을 복사
01D1  26: 8B 1E 0022                       mov    bx, es:[0022h]               ; 8번 인터럽트 벡터 세그먼트를 복사
01D6  26: 8B 0E 0024                       mov    cx, es:[0024h]               ; 9번 인터럽트 벡터 오프셋을 복사
01DB  26: 8B 16 0026                       mov    dx, es:[0026h]               ; 9번 인터럽트 벡터 세그먼트를 복사
01E0  A3 0103 R                            mov    word ptr Old8Int, ax         ;
01E3  89 1E 0105 R                         mov    word ptr Old8Int+2, bx       ; 각각 복사해둔 인터럽트 벡터
01E7  89 0E 0107 R                         mov    word ptr Old9Int, cx         ; 들을 마련해둔 장소에 저장한다.
01EB  89 16 0109 R                         mov    word ptr Old9Int+2, dx       ;

                                  ; 원래의 인터럽트를 새로운 인터럽트들로 바꾸어 놓는다.

01EF  FA                                   cli                                 ; 인터럽트를 불가능하게 한다.
01F0  8D 06 0174 R                         lea    ax, NewTimer                 ; 새로 정의한 타이머를 지목한다.
01F4  26: A3 0020                          mov    es:[0020h], ax               ; 오프셋과 세그먼트를
01F8  26: 8C 0E 0022                       mov    es:[0022h], cs               ; 새로운 타이머로 바꾼다.
01FD  8D 06 0111 R                         lea    ax, NewKbd                   ; 새로 정의한 키보드를 지목한다.
0201  26: A3 0024                          mov    es:[0024h], ax               ; 오프셋과 세그먼트를
0205  26: 8C 0E 0026                       mov    es:[0026h], cs               ; 새로운 키보드로 바꾼다.
020A  FB                                   sti                                 ; 인터럽트를 가능하게 한다.

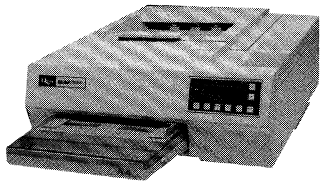
                                  ; 메모리에 남겨 두고 프로그램을 끝낸다.

020B  8D 16 01C9 R                         lea    dx, install                  ; 상주루틴 끝을 지목한다.
020F  CD 27                                int    27h                          ; 메모리에 남겨두고 끝낸다.
0211                              install  endp

0211                              code   ends
                                         end    start
```

신제품

LCD 디스플레이가 부착된 레이저 프린터
QLBP 3000



QLBP3000은 콘트롤러를 자체에 내장하여 크기를 기존의 프린터에 비하여 대폭 축소시켰으며 CPU로 MC68000을 채용하였다. 그리고 미국 휴렛패커드사의 HP 레조트모드를 100% 지원함으로써 레이저 프린터를 사용하는 각종 영문, 그래픽소프트웨어의 출력속도를 향상시켰으며 국가표준코드를 사용하여 행정 전산망용 표준 소프트웨어도 지원할 수 있도록 설계돼 있다.

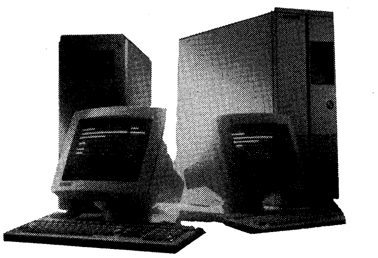
또한 자동 페이지정리기능이 있어 인자후 다시 정리할 필요가 없이 자동적으로 정리가 되고 토너가 카트리지 형식으로 되어 있어 토너의 교환이 용이한 레이저프린터이다.

제품의 규격은 다음과 같다.

- CPU : MC68000
- 형식 : 탁상형, 페이지 프린터
- 프린트방식 : 반도체레이저 건식전자사진방식
- 프린트속도 : 분당 최대8매(표준용지)
- 해상도 : 300dpi
- 사용문자 : 한글(명조, 고딕), 한자, 영문자
- 페이지당 문자수 : 80×62, 106×48
- 프린트기능 : 밑줄, 역상, 확대기능 (가로2배, 세로2배)
- 그래픽기능 : 풀 그래픽
- 칼라 : 카트리지 교환방식(단색)
- 인터페이스 : 패러럴, 시리얼

문의처 : (주)큐닉스
전 화 : (02)783-4301

중소기업 전산화에 적합한 미니컴퓨터
S/480



이 제품은 컨버전트사의 제품인 S시리즈 중형모델로 기존의 S시리즈와 호환성있게 설계된 MC 68020 프로세서, 유닉스시스템 V를 O/S로 채택한 32비트 미니컴퓨터이다.

또한 S/480은 하이테크 아키텍처로 설계된 CPU와 고속전송 속도를 가지는 SCSI를 사용하여 최대 96대의 단말기를 부착할 수 있고, VME BUS 사용시에는 대용량 SMD의 사용도 가능하다.

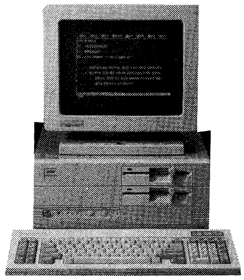
S/W는 시리즈간 100% 호환성이 있으며 표준통신 프로토콜을 이용하여 네트워크 형성이 용이하다. 가격에 비하여 성능이 뛰어나 중소기업의 전산화에 알맞은 시스템이다.

제품의 규격은 다음과 같다.

- CPU : MC68020 25MHz
- 캐쉬메모리 : 128KB
- 버츄얼메모리 : 1GB
- 메인메모리 : 4, 8, 16MB
- 메모리버스 : 64비트
- 시리얼포트 : 32개
- 패러럴포트 : 2개
- 보조기억장치 용량 : 80MB, 145MB, 325MB

문의처 : 경한시스템
전 화 : (02)784-7731

보급형 PC
JUMBO XT-1000



JUMBO XT-1000은 주기억용량이 256KB로 최대 640KB까지 메인보드상에서 확장이 가능하고, CPU를 8088-1을 채용해 클럭속도가 4.77/8/10MHz로 변환이 가능하며 리얼타임 클럭보드를 내장하고 있다.

또한 RS-232C 카드가 기본적으로 내장되어 있어 별도의 통신카드 부착없이도 통신이 가능하다. 한글통신도 가능하며 데이터통신망을 이용한 텔렉스 대용으로도 사용이 가능하다.

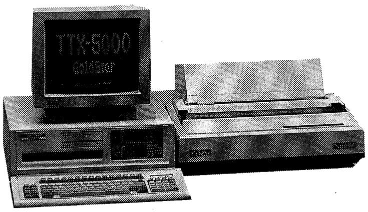
사용용도는 개인용, 교육용은 물론 사무자동화, 행정망, 터미널 등에 이용될 수 있다.

이 제품의 규격은 다음과 같다.

- CPU : i8088-1
- 속도 : 4.77 / 8 / 10MHz
- 주기억용량 : 256KB
- 보조기억장치 FDD : 1대
- 시리얼포트 : 1개
- 패러럴포트 : 1개
- 비디오보드 : MGA
- 슬롯 : 5개

문의처 : 동양시스템
전 화 : (02)587-2541~8

워드프로세서 기능의 텔리텍스
TTX-5000



이 텔리텍스는 체신부 표준규격을 적용한 제품으로 텔렉스기능에 문서편집이 가능한 워드프로세서기능이 추가된 통신기기로서 전용 네트워크가 아닌 일반전화선에 연결하여 사용할 수 있고, 기존에 설치되어 있는 사설교환기와 연결하여 사용이 가능하다.

그리고 이 제품은 문자와 그래픽을 혼용하여 한화면에 동시처리할 수 있을 뿐아니라 각종 도형의 제작, 특수문자, 겹쳐쓰기, 확대/축소 등 문서를 작성할 수 있는 다양한 그래픽기능을 갖추고 있다.

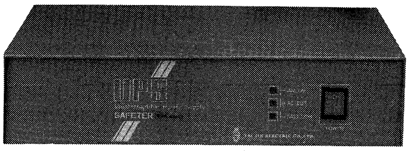
또한 전원오프 상태에서도 수신신호를 감지하여 자동으로 수신이 가능한 자동파워기능, 사람이 없어도 원하는 시간에 원하는 장소로 전송이 가능한 통신예약기능, 송신예약을 쉽게할 수 있도록 20개 번호까지 단축하여 사용이 가능한 단축다이얼기능, 한번의 조작으로 6명의 가입자까지 순차적으로 문서전송이 가능한 순차동보기능 등을 비롯하여 CF(Conversion Facility)를 통해 텔렉스와 통신이 가능하며 데이터 네트워크에도 접속이 가능한 다목적 통신장비이다.

제품의 규격은 다음과 같다.

- CPU : I8088
- 보조기억장치 : 5.25인치 FDD 2대
- 영상제어 : 한글 / 영문자 / 한자 / 그래픽
- 해상도 : 960×480
- 사용회선 : 공중전화 교환망, 구내전화 교환망, 데이터망
- 통신속도 : 2400, 1200, 600bps
- 모뎀방식 : V.22A, V.22

문의처 : 금성통신
전 화 : (02)787-4017

정현파 UPS
SO371B



이 제품은 평상시에는 노이즈를 제거한 양질의 전원이 공급되고 있다가 ±5%의 입력변동이 있으면 자동으로 완전정현파의 전압을 공급한다. 또한 순간정전, 심한 전원장해로부터 시스템을 안전하게 보호한다.

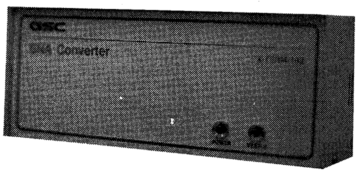
이 제품은 소음이 거의 없으며 정전시에는 경보음이 울리고, 자동충전회로의 내장으로 자동충전이 가능하다. 사용용도는 사무자동화나 공장자동화, 의료기기 등 여러분야에 사용이 가능하다.

주요 특징은 다음과 같다.

- 용량 : 0.3KVA
- 정전보상시간 : 5 ~10분

문의처 : 태진전기
전 화 : (02)277-5068

IBM통신 프로토콜 변환장비
SNA 콘버터



이 제품은 IBM 컴퓨터 터미널 및 프린터 사용에 있어, 고가의 IBM5550/40 PC 및 ITT 단말기를 대신하여 비동기(ASYNC) 터미널 및 프린터를 연결함으로써 IBM 컴퓨터 네트워크를 경제적으로 구성할 수 있는 IBM 통신, 프로토콜 변환장비이다.

SNA 콘버터를 이용할 경우 유저가 각종 비동기(ASYNC) 방식의 터미널 및 프린터를 IBM 컴퓨터에 연결할 수 있게 되어 기존 비용의 1/3수준으로 IBM 네트워크를 구성할 수 있다.

문의처 : 금성전선
전 화 : (02)787-2346/2341

워드프로세서
으뜸글 1.0



으뜸글 1.0은 화면상의 편집상태를 그대로 프린터로 출력하는 것이 가능하고 다양한 크기의 글씨체를 제공하고 있으며, 편집중의 이미지 뿐만아니라 인쇄시 용지 이미지도 표시되므로 출력장치의 자유로운 조절이 가능하다.

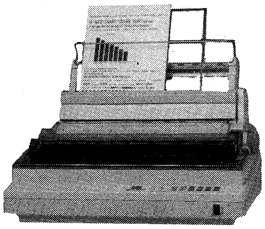
또한 화면분할 기능, 편리한 메뉴조작과 화일관리 기능, 중요부분의 강조기능(밑줄, 망, 선) 글자열의 좌우 중앙정돈기능, 매크로기능, EGA칼라기능, 작업환경의 설정기능, 사용자의 정의문자 지원기능 등이 제공되고 있으

며 또한 한자사전을 제공하여 한자처리가 가능한 워드프로세서이다.

IBM-PC XT/AT 호환기종 및 옴니스테이션시리즈에 모두 사용이 가능하며, 한글/한자 MS-DOS 버전 3.20 (KSSM 코드체계) 이상에서 수행한다.

문의처 : 큐닉스
전 화 : (02)783-4301

24핀 도트 매트릭스방식의 고속 프린터
OPT-260



이 제품은 다양한 한글, 한자코드를 선택할 수 있고, 2바이트 조합형, 완성형, N바이트, 행정전산망 표준 코드를 지원하는 프린터이다.

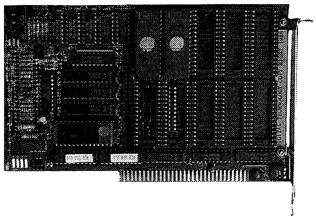
그리고 PC의 어떤 기종에도 상관없이 접속이 가능하고 중·대형컴퓨터에도 사용할 수 있으며 4색 칼라리본에 의해 7가지의 색상을 표현할수 있다.

제품의 규격은 다음과 같다.

- 프린트방식 : 임팩트 도트매트릭스
- 프린트속도 한글, 한자 : 174CPS
  고속영문 : 260CPS
- 문자구성 한글, 한자 : 22×22
  고속영문 : 24×13
- 행당 최대문자수 영문자 : 180자
  한글 : 90자
- 복사매수 : 원본1매+4매
- 문자코드 : 2바이트 완성형 및 조합형 IBM 2바이트 조합형
- 접속가능 시스템 : PC(XT/AT/386), PS/2 IBM5550/5549, CRT 터미널, DPS, VAX, 프라임

문의처 : 윈컴퓨터
전 화 : (02)556-5265

다기능으로 사용이 가능한 롬 디스크
PCE/PG 180K



이 제품은 FA나 CAD/CAM용 워크스테이션 엔지니어들을 위한 것으로 먼지와 진동에 강한 롬 디스크이다.

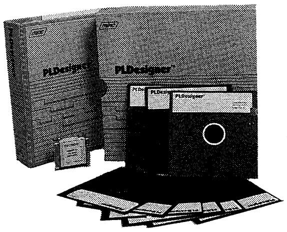
동 제품의 특징은 표준 디스크 드라이브와 에뮬레이트되고 자동부팅이 가능하며, 제3의 드라이브로도 사용할 수 있다. 도스가 내장된 훰웨어로 S/W 적으로 변환할 수 있고, I/O 모드를 프로그램화하여 DMA 채널을 사용한다. 또한 기존의 HDD에 비해 100배까지 속도를 증가시킬 수 있다.

응용 가능한 분야는 디스크가 없는 시스템과 고속·고신뢰성의 시스템, 산업 제어시스템, 공장 점검시스템, CAD/CAM용 워크스테이션, DOS, LAN, 워드프로세서, 터미널 에뮬레이터, 네트워크 시스템 등 여러 분야에 사용이 가능하다.

운용 환경으로는 XT, AT, PS/2의 드라이브(180K~1.44MB)와 에뮬레이트할 수 있고 한 컴퓨터에서 5MB까지 사용하며, I/O 인터페이스가 가능하다. 데이터를 전송할 때 DMA 채널 1을 사용하며, I/O 매핑으로 램 어드레스 공간없이도 메모리 용량을 증가시킬 수 있다.

문의처 : 디지탈 테크놀러지
전 화 : (02)581-1511

로직 디자인툴 패키지
PLDesigner



PLDesigner는 PLA, PLE, CAL, PAL, PLS, PROM 등의 로직 IC에 엔지니어가 필요로 하는 로직기능들을 컴퓨터를 사용해서 디자인하고, 컴파일링하여 프로그래밍할 수 있는 로직 디자인툴 패키지이다.

이 제품은 유저가 원하는 내용을 설계한 후 입력방법에 의해 입력하고 시뮬레이션, IC의 선택, 오토매틱 시뮬레이션 과정을 거친 후 새로운 로직IC를 만들어낸다. 그리고 자체에 메뉴형식의 편리한 워드프로세서를 가지고 있으며, 각종 워드프로세서를 PLDesigner 상에서 그대로 사용할 수 있다.

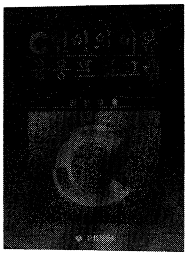
자체에 온라인 Help기능을 갖추고 있어 어떠한 명령어에 대해서도 즉시 도움말의 내용을 볼 수가 있고, 사용하기 쉬운 팝업 메뉴방식으로 설계되어 있는 메모리 소자 디자인 전용 패키지이다.

문의처 : 제우스컴퓨터
전 화 : (02)784-7841

신간안내

C언어의 이론 응용프로그램

강성수 저/성안당 발행/
정가 7,000원/525면/4·6배판



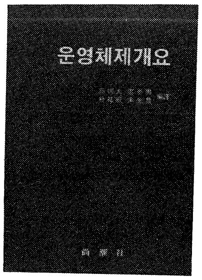
C언어는 하이레벨 언어의 장점 뿐만 아니라 비트단위 조작 등의 여러가지 로우레벨 언어의 장점까지도 두루 갖추고 있는 언어이다.

이 책은 C언어를 처음 접하는 독자들을 위하여 짧은 프로그램들을 게재하여 독자들이 여러 각도로 분석할 수 있도록 고려하였다. 그리고 표준함수에 관하여 전분야의 내용을 각 기능별로 요약 정리하여 보다 C언어를 효과적으로 이해하고 응용할 수 있도록 하였다.

또한 부록에는 미닉스 소스리스트가 소개되었는데 C언어를 학습하려는 사람들에게 많은 도움이 될 것이다.

운영체제 개요

고형대외 3인 편역/상조사 발행/
정가 5,500원/296면/4·6배판



국내에 출간된 운용체제에 관한 교재는 일반적으로 상당히 많은 기초지식을 요구하며, 내용면에서는 운영체제를 사용하는 측면보다는 설계하는 측면에서 고려되어야 할 이론적인 기술이 많이 설명되었다.

이 책은 컴퓨터를 공부하고자 하는 학생들에게 지나친 이론에 치우치기 보다는 어디서나 쉽게 접할 수 있는 실제 시스템(MS-DOS, UNIX)을 예로 들어 실습과 함께 운영체제의 기초 개념을 배울 수 있도록 작성되었다.

이론적인 면 보다는 실제 시스템적인 면에 촛점을 맞추어 다양한 운영체제의 기능과 이론을 설명하였으며, 상세하고 구체적인 방법에도 접근하고 있다.

각 장의 끝에는 그 장에서 주요 내용 혹은 주의를 기울여야 할 내용을 간략하게 기술하였다.

우리들 주변에서 흔히 접할 수 있는 운영체제들의 시스템적 관점에서 운영체제에 대해 이해할 수 있도록 기술한 서적이다.

RM/COBOL 프로그래밍

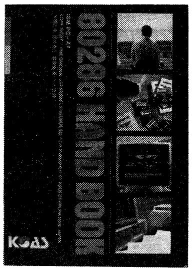
이태욱외 2인 공편저/교학사 발행/
정가 4,500원/263면/4·6배판



이 책은 종래 메인 프레임에서 사용된 코볼을 벗어나 퍼스널컴퓨터에 알맞고 다양한 응용성을 갖고 있는 RM/COBOL을 소개한 서적이다. 또한 RM/COBOL의 개괄적인 내용뿐만 아니라 사무자동화에 필요한 주변환경까지 다루고 있는 것이 특징이다. 그리고 RM/COBOL 운영시스템 종류와 환경을 설명하여 RM/COBOL의 사용자에게 많은 도움을 줄 수 있도록 구성하였고 사용문자와 프로그램 구성체계 등을 설명함으로써 내용을 이해하기가 편리하도록 배려하였다.

80286 HANDBOOK

천광호 역/코아스 발행/
정가 7,500원/383면/4·6배판



80286 CPU는 과거에는 대형컴퓨터만 실현시킬 수 있었던 메모리 관리, 보호기구, Task관리 등의 기능을 1칩에 집적시켜 놓은 것이다.

이 책은 80286 CPU의 어플리케이션 또는 시스템 프로그램에 관한 유저를 대상으로 명령세트, 어셈블러 프로그래밍에 관해 구체적인 사례를 들어 설명하고 있다.

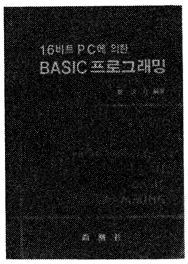
내용을 살펴보면 제1부 80286 아키텍처는 CPU의 특징, 레지스터구성, 세그먼트의 개념 등을 설명하고, 실제 어드레스 모드의 기능에 대해 메모리 관리, 예외처리, 초기화 등을 기술하고 있다. 제2부에서는 명령어에 대하여 기술하였는데 알파벳 순서가 아닌 기능에 따라 분류하여 설명하고 있어 한눈에 명령어를 비교할 수 있게 구성되어 있다. 제3부에는 어셈블러 프로그래밍의 설명으로 실제 어드레스 모드와 프로텍트모드에 있어서 시스템 구성상의 차이에 따른 어셈블러 프로그래밍의 차이에 대하여도 설명하였다.

16비트 PC에 의한 베이직 프로그래밍

정문재 편저/상조사 발행/
정가 6,000원/336면/4·6배판

이 책은 전산학을 처음 접하는 사용자를 대상으로 하고 있는데 16비트 PC를 이용한 베이직언어를 숙지하는데 주안점을 두고 있다.

전산학의 기초적인 설명으로 수학에 대한 기초내용이 소개되어 시스템의 정보 표현방식을

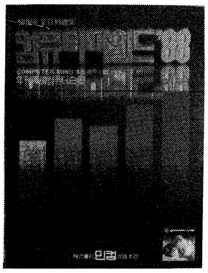


이해하는데 도움이 되게끔 하였으며 16비트 컴퓨터를 이용한 프로그래밍 작성방법을 예제를 통하여 설명하고 독자들의 이해의 폭을 넓히고 있다.

베이직의 기본 프로그램 및 응용 프로그램에 관한 문제를 유형별로 흐름도와 함께 제시하여 프로그램의 개념을 빨리 파악할 수 있도록 배려한 것이 특징이다.

컴퓨터마인드 '88

민컴 편집부/민컴 출판본부 발행/
정가 10,000원/777면/4·6배판



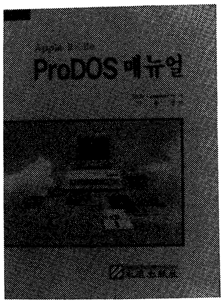
정보는 미래지향적이고 새로운 사회상을 제시해 주는 선도자일 뿐만 아니라 변혁을 주도하는 구심점이기 때문에 누구든지 수집 발굴하여 이용할 수 있다. 이러한 취지에서 발간된 이 책은 각종 최신자료를 바탕으로 게재내용의 밀도와 넓이를 대폭 강화하여 일목요연하고도 체계적으로 정리 편집하여 국내외 정보 산업관련 모든 분야에 충실한 가이드 역할을 하도록 하였다.

이 책의 구성은 전체 6개의 장으로 되어 있는데 4장까지는 주로 국내 산업현황과 국내 컴퓨터 이용현황, 메이커별 컴퓨터 및 소프트웨어 공급현황 등이 소개되어 있으며, 그밖에도 국내 컴퓨터 관련 H/W, S/W, 주변기기 회사 등을 총망라하였다. 그리고 5장과 6장에서는 해외 정보산업 현황과 정보산업 관련자료를 소개하고 있다.

정보산업에 종사자나 관련업체에 관심이 있는 사람이면 한번쯤 읽어야 하는 컴퓨터 산업 전반에 관한 정보지이다.

ProDOS 매뉴얼

김용성 역/영진출판사 발행/
정가 3,500원/159면/4·6배판



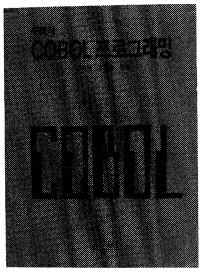
애플 사용자를 위하여 새로이 개발된 ProDOS (Profession Disk Operating System)는 종래의 Apple II DOS3.3보다 기능이 보강된 OS이다.

이러한 ProDOS의 매뉴얼로 작성된 이 책은 3편으로 구성되었다. 제1편에서는 ProDOS 유저 디스크를 소개하고, 제2편에서는 ProDOS 화일에 대하여 상세히 설명하고 있는데, 개요와 계층적 화일구조와 패스명의 사용법을 설명하였다. 그리고 DOS-ProDOS 변환 프로그램에 대해서도 설명하고 있다.

부록에서는 ProDOS 명령 실행시에 발생할 수 있는 각종 에러메시지를 열거하고 그 원인과 조치를 간략하게 기술하여 사용자들이 참조할 수 있도록 배려하였다. 또한 초보자들을 위하여 용어해설까지 기술한 매뉴얼이다.

구조적 COBOL 프로그래밍

김하진·류성열 공저/생능출판사 발행/
정가 7,900원/536면/4·6배판



코볼의 자료처리 기법은 하드웨어의 발달과 프로그래밍 언어의 발전에 따라 변화를 거듭하여 왔다.

구조적 프로그램의 설계는 제시된 문제점을 해결하기에는 좋은 방법론이긴 하지만 코볼을 처음 배우고자 하는 사람들에게 체계적으로 교육하거나 자습하기에는 부족한 점이 많았다.

이 책은 코볼을 처음 배우는 사람이 프로그램을 효율적으로 설계하고 코딩할 수 있는 구조적인 방법론과 관련문법을 익힐 수 있도록 작성하였다. 또한 이 책은 코볼에서 가장 중요한 화일 처리기법을 기본적인 이론과 아울러 실제적인 처리방법에 중점을 두어 실무적응력향상에 비중을 두고 있다.

이 책의 구성은 기초편과 응용편으로 구분되어 있는데 기초편에서는 컴퓨터에 관한 기본적인 이해와 코볼프로그램에 관한 기초지식을 교육하거나 자습할 수 있도록 하기 위하여 컴퓨터 프로그램과 프로그래밍 언어의 관계를 이해하고, 좋은 프로그램을 설계할 수 있도록 코볼언어의 이론적 배경을 위주로 설명하고 있다.

응용편에서는 일반적인 자료처리 업무를 수행하기 위한 실무능력을 배양하기 위한 단원으로서 순차화일, 인덱스 화일, 상대적 화일, 화일의 분류와 검색 및 테이블 처리 등을 관련문법과 예제를 중심으로 설명하였고, 특수용법으로 한글 자료처리, 화면 설계방법, 에러 처리용법 및 레포트 작성의 사용법에 대하여 기술하고 있다. PC

새롭게 등장한 미래 지향적인 컴퓨터 POWER 88

컴퓨터가 대중화되면서 각 기업들이 새로운 제품들을 계속해서 발표하는 등 컴퓨터 시장은 활기를 띄고 있다. 또한, 정부의 '행정전산망 구축'으로 인하여, 그 열기는 한층더 고조되어 가는 느낌이다.

특히, 현대전자에서 지난해 출고한 Supper 16E는 보급형 PC로서 16비트 XT기종중 가격면에서 종래의 인식을 바꾸어 놓을 만큼 커다란 반응을 불러 일으켰다. 교육용에 적합한 동 기종은 컴퓨터마인드 확산에 하나의 기폭제가 되었다고 볼 수 있다.

현대전자가 국민보급형 PC로 수퍼 16E를 출하한 이래 삼성, 금성, 대우, 삼보 등에서도 잇달아 이와 유사한 제품을 출하하고 있다.

반면에, 뒤늦게 국내 PC시장에 뛰어든 효성은 국내 16비트 PC/XT 기종들이 일반적으로 5.25인치 FDD를 선택하고 있는데 비해, Power 88이란 제품을 선보이면서 3.5인치 FDD를 부착할 수 있게 하였다.

3.5인치 FDD는 메모리 용량, 데이터 보관 등 여러가지 면에서 5.25인치 FDD보다 우수하여 세계적인 추세가 차세대의 주력 보조기억장치로 자리를 잡아가고 있는 실정이다.

현재 파워88과 유사한 제품으로는 진컴퓨터의 'Superset-30', 대우통신의 '프로-엘리트' 등을 들 수 있다.

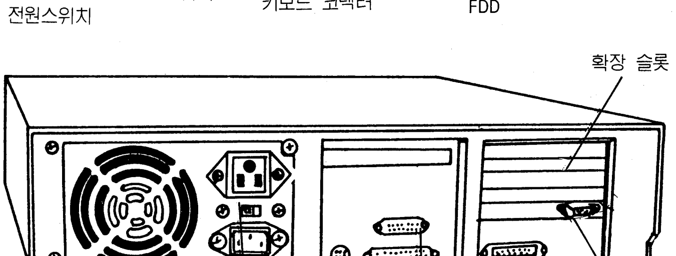
구성

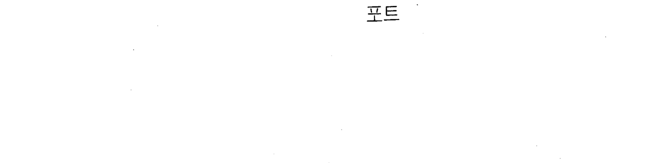
컴팩트한 슬림형인 파워88은 외관부터 구매자의 눈길을 끌기 충분하리 만큼 깔끔하게 나온 제품이며 무엇보다도 작은 공간에서 사용하는데 무리가 없을 만큼 크기가 작다(본체의 높이가 불과 99㎜ 밖에 되지 않는다.).

사진 1에서 보는 것처럼 시스템 본체의 우측에는 5.25 인치 FDD와 3.5인치 FDD가 설치되어 있으며, 우측하단에는 HDD를 설치하기 위한 공간이 마련되어 있다. 또한 전면에 전원스위치, 리세트스위치, 키보드코넥터등이 있다.

그리고 본체 후면에는 패러럴포트, 비디오포트, 시리얼포트, 게임포트, 마우스포트, 전원콘센트 등이 나와있다.

〔사진 1〕 파워88의 외관





파워88의 기본구성은 표 1에서 보는 바와 같은 표준 XT기준에 준하며 BIOS는 Award Software의 ROM BIOS를 사용하고 있고, 다른 기종과는 달리 3.5인치 FDD (1.44MB) 1대를 부착하고 있는 것이 특징이다.

옵션으로는 HDD/HDC, 한글/한자카드, 시리얼포트 1개, 8087연산 보조프로세서 등을 각자 용도에 맞게 선택할 수 있다.

CPU속도선택은 다른 컴퓨터와는 달리 콘트롤키로서 하게 되어 있다.

처음 컴퓨터의 전원을 'ON'시키면 클럭속도는 자동으로 9.54MHz로 설정

된다. 여기서 만일 4.77MHz로 변환하려면, 키보드상에서 〈Ctrl〉키와 〈Alt〉키를 동시에 누른 상태에서 숫자 키패드(Numeric Keypad)의 '–'키를 누르면, 두번의 '삑'소리가 나고 4.77MHz로 변환된다. 다시 4.77MHz로부터 9.54MHz로 변환시키려면, 〈Alt〉키와 〈Ctrl〉키 그리고 숫자 키패드 '+'를 동시에 누르면 한번의 '삑'소리가 나며 변환된다.

또한, 칼라/그래픽(EGA카드)을 지원하는 비디오 보드가 내장되어 있어 별도의 비디오 보드를 설치할 필요가 없다. 만일 다른 종류의 비디오 보드를 사용하고자 한다면, 현재 설치된 어댑터를 제거하고 새로운 어댑터를 설치하면 된다.

시스템의 특징

IBM-PC 호환기종이면 시스템 구조가 같듯이 효성 파워88에서 굳이 기존의 다른 기종과 차이점을 꼽는다면 3.5인치 FDD를 장착한 점을 들 수 있다. 이외에도 플로피디스크를 제어하는 FDC(37C65)가 카드로 되어 있지 않고 단일 칩으로 마더보드에 위치하고 있는 점도 기존의 기종들과 크게 다른 점이라 할 수 있다. 이는 보드자체의 면적을 줄이고 동작의 신뢰성을 높이기 위해서 그렇게 구성되었다고 한다.

여기서는 마더보드의 고집적성, 메모리의 확장, 3.5인치 FDD의 기능 그리고 자체진단 기능 등을 알아본다.

●마더보드의 고집적성

사진 2, 3에서 보는 바와 같이 커스텀 IC 등을 사용하여 마더보드의 부피를 대폭 축소시켰으며 입출력 포트를 별도로 장착할 필요없이 I/O 포트가 백보드에 직접 부착되어 있다. 또한 확장 슬롯이 보드상에 직접 부착되어 있는 것이 아니라 마더보드상에는 1개의 슬롯만이 있으며 여기에 확장카드를 꽂아 주변기기 확장에 필요한 5개의 슬롯을 구성할 수 있다(사진 3 참조).

그림 1의 블록다이어그램에서 보면 중앙부분의 제어선을 기준으로 좌측은 시스템부를 가리키며, 우측부분은 I/O부을 나타내고 있다.

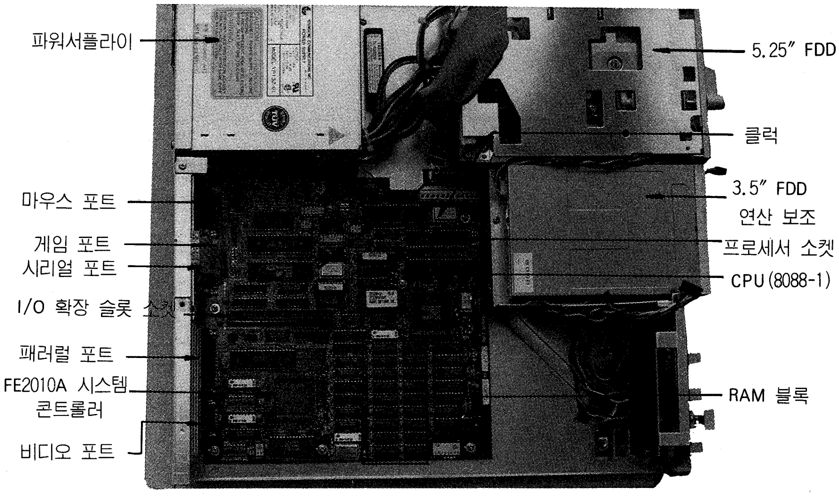
부품자체를 커스텀 IC를 사용하여 마더보드의 크기를 줄이고, 효율적인 부품배치를 하였으며 부품의 갯수도 대폭적으로 줄어 들었다.

즉, FE2010A(Western Digital사 제품) 시스템 콘트롤러는 8259(인터럽트

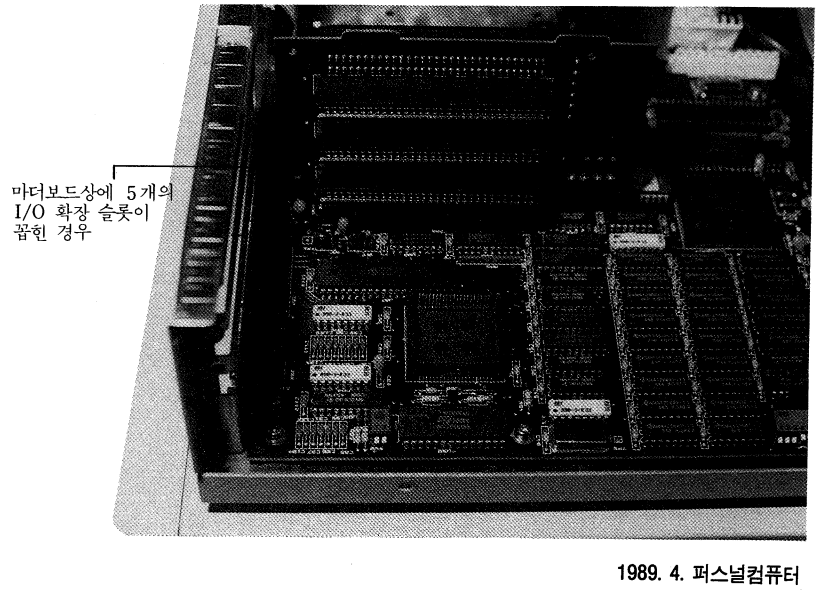
〔표 1〕 기본 구성

CPU		인텔 8088-1
클럭속도		9.54/4.77MHz
주기억장치	ROM	16KB
	RAM	640KB
보조기억장치	FDD	5.25″(360KB 1개)
		3.5″(1.44MB 1개)
	HDD	32/65MB(옵션)
내장기구		시리얼 포트 : 1개 패러럴 포트 : 1개 I/O 확장 슬롯 : 5개 마우스 및 게임 포트 리얼타임클럭

〔사진 2〕 마더보드



〔사진 3〕 I/O 확장 슬롯이 꼽힌 마더보드



콘트롤러), 8237(DMA 콘트롤러), 8253(인터벌 타이머), 8284(클럭 발생과 드라이버), 8288(버스 콘트롤러), 8255(패러럴 I/O) 등을 커스텀화하여 하나의 칩으로 만든 것이다.

또한, 37C65(FDC : 5.25, 3.5인치 FDD를 제어한다.), 82C50(UART라 불리우고 RS-232C를 지원하며 시리얼통신에 사용), 82C11(패러럴포트를 지원하며 주로 통신포트를 지원하는데 사용), 58167(RTC : 리얼타임 클럭을 발생), 그리고 TC6010G(Twin Head사 제품으로 마우스와 게임을 지원) 등도 커스텀화 하였다.

●메모리의 확장

기본적으로 640KB의 주 메모리를 지원하고 있다. 이 이상의 주 메모리 영역을 필요로 할 경우에는 보통 XT 시스템에서와 같이 확장메모리 카드를 I/O 슬롯에 장착하고 EMS를 세팅하면 된다.

이때 주의할 점은 점퍼의 위치를 변경하여야 하고, 확장메모리 보드의 DRAM 은 150nsec 이하가 되어야 한다.(예를 들면 TI41256-15나 MCM6256 AP 15, 또는 그 유사제품을 사용). 그 이유는 시스템 클럭을 DRAM이 액세스하는 시간을 충분히 서포트하기 위해서 이다.

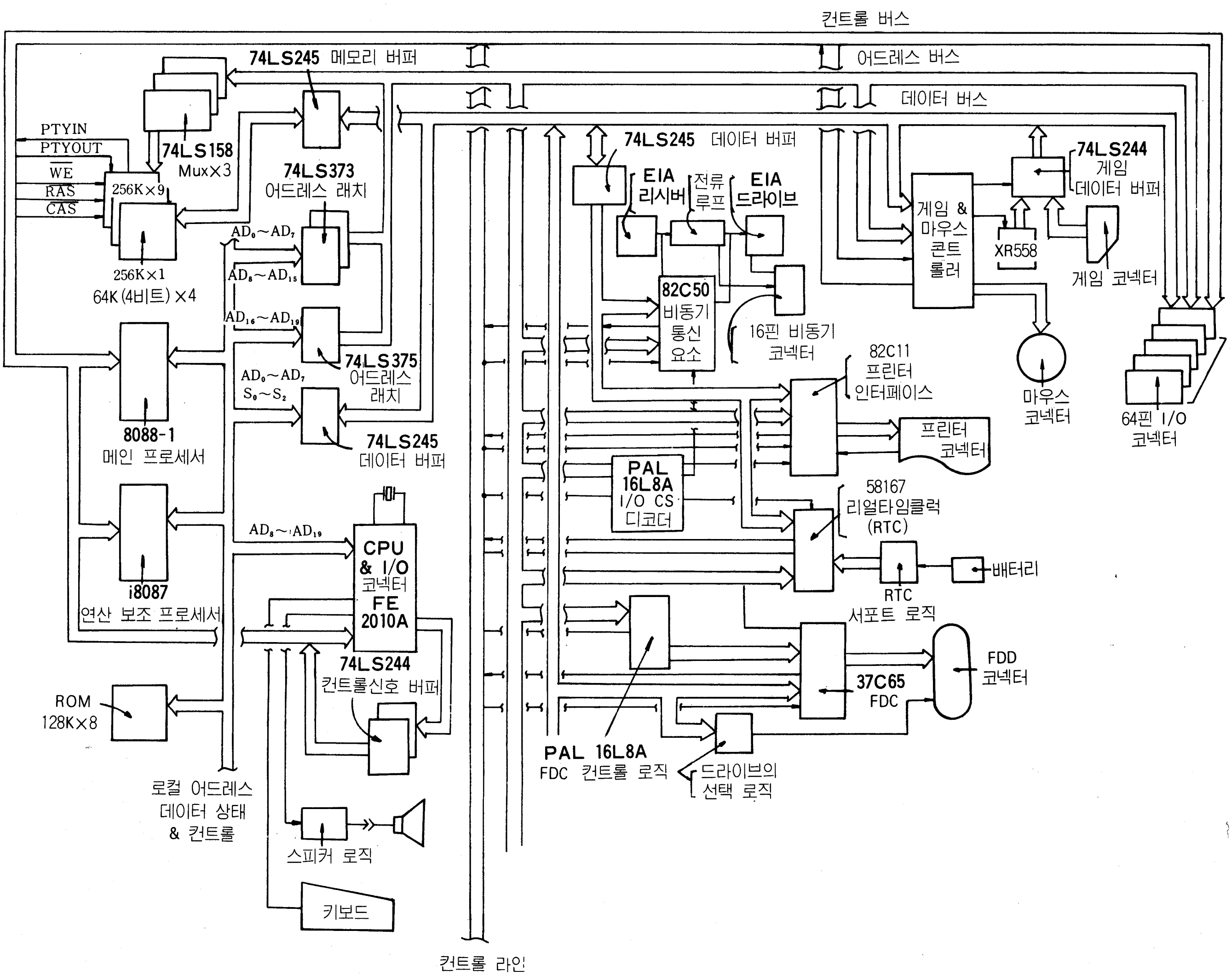
●3.5" FDD

PC시장의 FDD사용이 3.5인치를 사용하는 추세로 변화하고 있고, 구미의 경우에는 FDD의 사용중 60%가 3.5인치 FDD 타입이다.

3.5인치 FDD는 사이즈가 컴팩트하고, 보호자켓이 튼튼하며 충격에 강하므로 깨어질 염려가 적고, 보관이 간편하다는 등의 여러가지 장점을 지니고 있다.

3.5인치 FDD를 지원하기 위해서는 FDD용 케이블에 3.5인치 드라이브용

〔그림 1〕 블록다이어그램



코넥터가 부착되어야 하고 BIOS에서의 지원이 필요하며 3.5인치 드라이브를 위한 별도의 FDC는 필요하지 않으며 단지 딥스위치의 2번핀을 선택하기만 하면 된다.

HDD의 경우에는 별도로 HDC를 슬롯에 장착하여야 되고, 3.5인치 FDD 대신에 5.25인치 FDD를 연결하여 사용할 수도 있다.(HDD가 설치되어 있을 경우에는 본체 외부에 따로 설치해야 함).

디스크 드라이브 지정방식이 파워88에서는 고정되어 있지 않고, 시스템이 부팅되는 드라이브가 드라이브 A:로 지정되도록 되어 있다. 즉 2대의 FDD 중 3.5인치 FDD로 먼저 시스템을 부팅하면 그 드라이브가 드라이브 A:로, 5.25인치 FDD가 먼저 부팅되면 그것이 드라이브 A:로 지정된다. 즉 서로 다른 데이터 테이블 값으로 ROM BIOS에서 이를 해결하고 있다.

또한, MS-DOS를 양쪽에 띄울 수 있어 편리하게 사용할 수 있다. 3.5인치 FDD는 용량을 대형화(1.44MB)하였고, XT/AT기종 구분없이 시스템 운용이 단순하고 간편하다. 또한, 한개의 디스켓에 많은 화일들을 등록하여 사용하는 것이 가능하므로 빈번한 디스켓 교체로 인한 번거러움을 피해 디스켓 사용에 효율성을 기할 수 있다.

●자체진단기능

파워88의 또다른 특징은 CPU, 램, 클럭속도 등을 전원을 'ON' 시킬때마다 확인할 수 있다는 점이다. 또한 효성컴퓨터에서 제공하는 한글 QAPLUS를 통해 유저가 현재 사용하고 있는 시스템의 각 부분에 이상이 있는지, 정확히 동작하고 있는지의 여부를 정확히 체크해준다.

구체적으로는 CPU 테스트로 기본기능, 수리명령, 논리명령, DMA제어, 인터럽트 등을 할 수 있으며, 램메모리 테스트는 기본 메모리와 확장 메모리로 나누어 테스트할 수 있다.

이외에도 키보드, 클럭/카렌다, 비디오 어댑터(MDA, CGA, EGA), 프린터(특정문자들의 프린트), FDD, HDD, 통신포트(내부 테스트는 통신제어창치 8251의 내부 룩백 특징을 점검하고, 외부 테스트는 RS-232C 접속 플러그를 구입하여 COM포트에 접속하여 체크함) 등을 테스트하는 것이 가능하다(화면 1 참조).

〔화면 1〕

```
필요한 테스트를 선택/해재하려면 지정하는 영문 키를 누르십시오. 선택한 다음, 테
스트를 시작하려면 스패이스키를, 중지하려면 Ctrl키와 'C'키를 동시에 누르십시오.
------------------------------------------------------------------------------
        Y  모든 테스트의 취소                          X  중복 테스트
        Z  모든 장치의 테스트
------------------------------------------------------------------------------
   ──── 주 요 장 치 ────                        ──── 하드디스크 ────
        A  CPU 테스트
        B  메모리 테스트
        C  키보드 테스트
                                                ──── 통신 Ports ────
                                                     Q  COM1 테스트

   ──── 비대오 어댑터 ────

        H  고해상도 모노 테스트
                                                ──── 프 린 터 ────
                                                     U  LPT1 테스트
   ──── 플로피디스크 ────
        J  플로피디스크 A: 테스트
        K  플로피디스크 B: 테스트
```

맺음말

효성 파워88은 처음에 수출형으로 나와 수출을 하다, 지난 1월 중순부터 국내에 판매되고 있다.

파워88의 가장 두드러진 특징은 앞에서도 언급했지만 컴팩트한 슬림형으로 외관이 뛰어나고, 3.5인치 FDD를 장착하고 있어 5.25인치나 3.5인치 FDD에 담긴 응용프로그램도 사용 가능하다는 점을 들 수 있다.

또한 커스텀 칩을 사용하여 고신뢰성과 동작에서 안정을 기했고, 이로 인하여 본체는 작지만 많은 기능을 갖고 있다. PC

완벽하게 회로를 디자인 해주는 만능 PCB용 캐드
DASOFT-16

오안식 · 전진수/중앙대 셈틀연구회

하드웨어를 제작하고자 할 때 항상 문제가 되는 것은 배선의 안정성 문제라고 볼 수 있다.

납으로 배선을 하다 보면 잡음에 영향을 받기 쉬워 완성하였다 하더라도 오동작을 일으킬 때가 많다. 이런 이유로 요즈음은 회로 설계를 비롯한 각종 하드웨어 제작에 캐드용 소프트웨어를 이용하는 사례가 많은 실정이다. 여기서 소개하는 다소프트 프로그램도 이런 성격의 소프트웨어로서 기판을 제작할 때의 문제점들을 시원스럽게 해결해 주리라 믿는다.

이 프로그램이 시중에 소개된 지는 꽤 된 것으로 알고 있으나 정작 유용하게 이용되는 사례는 적은 것 같다. 이는 매뉴얼 설명의 산만성과 PCB제작에 필요한 여러 장비들이 필요하다는 어려움 때문이라고 보며 이 글이 관심있는 독자들에게 유익한 조언과 방향제시가 되었으면 하는게 필자의 바램이다.

우선 프로그램을 실행해 보고 싶은 마음이 생기면 자연히 그에 따르는 행동이 나오는 것 같다. 한 예로 이 프로그램을 이용해서 실제로 기판을 제작하기 위해서는 플로터(Plotter)가 필요한데 필자는 대학원에 있는 플로터를 이용하기 위해 굉장한 노력을 한 적이 있다.

D.EXE의 실행

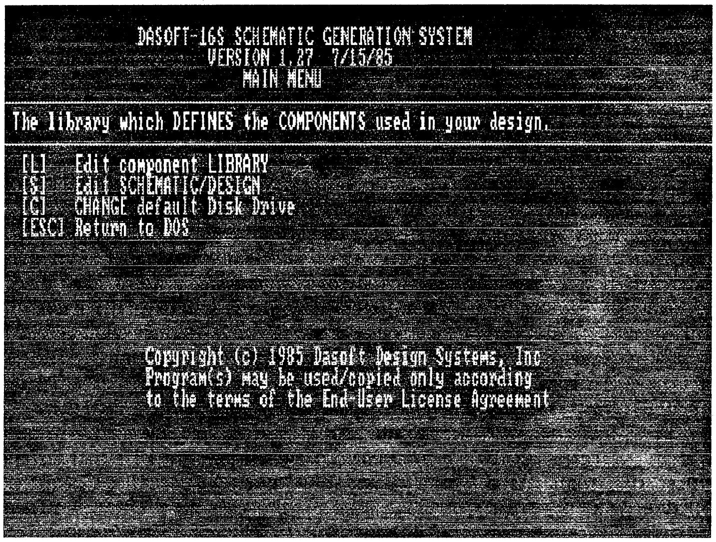
다소프트는 2개의 메인 프로그램, 즉 D.EXE(다소프트A)와 DB.EXE(다소프트B)를 비롯하여 기타 부품을 정의한 라이브러리(Library) 화일로 구성된다.

메인 프로그램을 A드라이브에, 라이브러리 프로그램은 B드라이브에 넣고 구동시키는데 D.EXE인 경우는 확장자가 DVC와 DES인 화일을 만들게 된다. 여기서 DVC(Component Library Editor)화일은 설계시 사용될 부품의 종류(IC는 핀수에 따라 그 중 하나를 정의하고, 저항이 여러개 쓰일 경우 핀수는 2개이므로 하나만 지정하면 된다.)를 작성하며 DES(Design File)은 실제 사용되는 부품의 수 만큼 정의하여 회로의 연결선을 정의하게 된다.

그 다음은 DB.EXE를 실행시켜 OTL과 LYT란 확장자를 가지는 화일을 만드는데 OTL(Board Outline Editor) 화일은 사용할 기판의 크기를 지정하고 LYT(Layout Editor)화일은 실제 기판에 그려질 배선을 정한다. 여기서는 실제로 사용 빈도수가 높은 부분을 중점적으로 설명하고 자주 쓰이지 않거나 실행시켜서 명확하지 않는 결과가 나타나는 부분은 생략했음을 밝혀둔다.

다소프트의 D.EXE부분에서는 메인 메뉴(화면 1 참조)에서 보는 바와 같이 크게 두 부분으로 나눌 수 있다. [L]은 디자인에 쓰일 각 구성소자들의 라이브러리 화일을 만드는 부분이고 [S]는 [L]에서 만든 구성소자들로서 도식적인 디자인을 작성하는 부분이다. 그리고 [C]는 디폴트 드라이브를 변경하

〔화면 1〕 D.EXE의 메인 메뉴



는 부분으로, DOS상태에서 'D'를 눌러 화면 1의 메인 메뉴로 들어온 다음에는 항상 [C]를 이용해서 작업 드라이브를 B드라이브 상태로 지정해 놓는 것이 좋다. [ESC]는 모든 작업 환경을 취소하고 이전의 메뉴 상태로 되돌아갈 때 쓰인다.

[L]을 입력해 들어가 보도록 하자. (선택 방법은 'L'을 직접 입력하는 방법과 상하 화살표키를 이용해서 하이라이트될 때 〈ENTER〉를 누르는 방법이 있다.) 사용자가 만들고자 하는 새로운 라이브러리 화일의 이름을 입력하면 다음과 같이 새로운 화일인가를 묻는 메시지가 표시된다.

```
This is a NEW file CREATE? (Y/N)
```

새로운 화일이면 'Y'를 입력하고 기존의 라이브러리 화일을 수정할 경우는 〈F9〉키를 누르면 되는데 이 경우는 다음과 같이 기존의 라이브러리 화일들이 화면에 디스플레이된다.

```
1 AD-CON.DVC
2 CIRCUIT.DVC
3 ANSIG.DVC
4 IOCARD.DVC
5 INTEL.DVC
6 SPARTS.DVC
7 DAC-0808.DVC
8 OAS.DVC
9 IBM-IO.DVC
```

임의의 숫자(여기서는 9번을 예로 하였다)를 선택한 후 〈ENTER〉키를 누르면 **화면 2**가 나타난다. 이 부분에서는 구성소자를 크게 능동소자(Active device 예 : IC, TR 등)와 수동소자(Passive device 예 : 저항, 콘덴서 등)로 구분을 하고 있는데 꼭 필요한 것은 아니지만 계속 작업을 하면서 구분을 해 놓으면 편리함을 알게 된다.

[A 와 [P]가 각각 능동소자와 수동소자를 정의하는 부분이고 [R]은 다른 라이브러리 화일로부터 소자를 읽어들이는 역할을 하고, [D]는 이미 정의되어 있는 소자를 삭제하는 부분이다. 그 밖에 [F]를 선택하면 기판에서 각 소자의 핀(Pin)이 갖는 모양과 간격을 정의하게 되고 [A]나 [P]에서 소자를 정의할 때 그 소자에 맞는 Footprint는 이미 [F]를 선택해서 작성해 둔 Footprint를 이용할 수도 있다. 그러므로 소자를 정의할 때 먼저 Footprint를 작성한 다음에 [A]나 [P]에서 소자를 정의하는 방법이 더 편리하다는 것을 알아두자.

'F'를 누른 후 이름을 입력한다. 기존에 있는 Footprint를 수정할 때는 〈F9〉키를 누른 후 다음 사항 중에서 원하는 번호를 선택하면 된다.

```
1 14PIN        F  14
2 16PIN        F  16
3 24PIN        F  24
4 2PIN-CAPA    F   2
5 30PINCONNE   F  30
6 34PIN-CONN   M  34
7 40PIN        F  40
8 FINGER-60    F  60
9 SLOT-16BIT   F  62
```

처음에는 화면 3의 상단 부분만이 표시되는데 이 상태에서 핀 수와 홀의 크기 및 패드의 모양을 지정하게 된다. 'P'를 눌러 핀 수로 14(74LS04)를 입력해 보자. 그리고 홀의 크기는 32, 패드의 크기는 60으로 선택하는 게 좋다(그림 1 참조). 패드의 모양은 여러가지가 있으나 보통 라운드(Round)형태를 사용한다. (필자의 경우 다른 모양을 선택해 보아도 라운드 형태로 밖에 나오지 않았다.) 이제 'L'를 누르면 **화면 3**의 전체가 화면에 나타났을 것이다. 여기서는 소자의 핀 위치에 따라 번호(1~7)를 선택한다. (예 : 저항-1번 SINGLE IN-LINE, 73LS04-2번 DUAL IN-LINE 등)그리고 열사이의 간격(Distance rows 예 : 74LS04-300, 8255-600 등)과 핀과 핀사이의 간격(Pin spacing : 대부분 100으로 한다)을 기입하고 나서 〈F1〉를 눌러 저장시키면 다시 화면 2로 돌아 간다.

그럼 직접 소자를 정의해 보기로 하

〔화면 2〕 라이브러리 화일(IBM-IO.DVC)

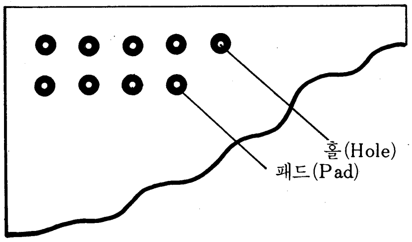
```
EDITING DEVICE FILE     IBM-IO.DVC
-----------------------------------------------------------
Parts NOT requiring a value at design time.
-----------------------------------------------------------
[A] Edit/Create  ACTIVE device      [P] Edit/Create PASSIVE device
[K] Edit the Key List               [F] Edit/Create FOOTPRINT
[L] LIST Entries by KEY             [D] DELETE any file Entry
[R] Read secondary library file
```

〔화면 3〕 16핀인 경우의 Foot Print 선택

```
FOOTPRINT 16PIN      #PINS  14  HOLE SIZE  32  PADSIZE  60  PAD  ROUND
-----------------------------------------------------------------------
 [P]ad & #pins    [L]ocate pins    [K]eys    [R]emarks  [F]ingers
-----------------------------------------------------------------------
[1-7] Select a STD    [RETURN] Map each Pin       [ESC] EXIT  without change.
-----------------------------------------------------------------------
     NUM.  PACKAGE STYLE

      1    SINGLE IN-LINE
      2    DUAL IN-LINE
      3    QUAD IN-LINE
      4    PINS 4 SIDES SQUARE
      5    PINS 4 SIDES RECTANGLE
      6    CIRCULAR
      7    TRIANGLE - 3 PIN ONLY
This footprint is a standard package of the type indicated above.
Distance in .001" between rows.    300
Pin spacing in .001"     100
```

〔그림 1〕 기판의 홀과 패드



자. 화면 2의 상태에서 [A]를 선택하면 소자의 이름을 입력받는 상태가 된다. 이 때 기존에 있던 소자를 수정할 때는 〈F9〉를 눌러 다음 사항중에서 원하는 번호를 선택한다. 이때는 **화면 4**가 나타나는데 'N'을 눌러 핀 수를 입력하고, 'P'를 눌러 핀 이름을 입력한다. 방법은 기입하고자 하는 핀번호 앞에 커서를 이동시키고 〈ENTER〉키를 누르면 이름을 입력할 수 있는 상태가 된다.

이 과정이 끝나면 오른쪽 하단에 있는 그림에서 알파벳(A~F)을 선택한다. 이것은 각 핀의 용도에 따라서 하나의 그룹을 만들어 놓음으로써 디자인할 때 편리하게 하기 위해서이다. (예를 들면 회로도에서 데이터 버스인 경우) 다시 〈ENTER〉키를 누르면 다음 핀번호 앞으로 커서가 이동한다. 이때 〈F1〉을 누르면 저장되면서 다시 화면 2의 상태로 돌아오는데 다른 작업 상황에서도 저장할 경우에는 항상 〈F1〉키를 이용하게 된다. ([P]인 경우도 [A]에서와 같은 형식을 취함)

이번에는 디자인하는 방법([S])에 대해 알아 보자. 화면 1 상태에서 'S'를 눌러 [L]에서와 마찬가지로 새로운 화일이름을 입력하고 확인 메시지에 대해 'Y'를 입력한다. 앞에서와 마찬가지로 기존의 디자인 화일을 수정할 때에는 〈F9〉를 눌러 해당 번호를 선택하면 된다.

화면 4가 나오면 먼저 〈F2〉를 눌러 해당 페이지를 선택(보통 1페이지를 선택)하여 활성화시킨다. 그러면 페이지의 크기를 묻는 메시지가 나타나는데 'A'가 가장 작고 'D'가 가장 크다. (보통 'D'를 선택)

〈F4〉키는 라이브러리 화일에 작성되어 있는 소자를 불러들여 페이지에 출력시키는 기능을 가지고 있는데 화면상에 소자를 찍을 때는 원하는 위치로 커서를 이동시킨 후 〈F4〉를 누르면 된다. 그 다음은 사용하고자 하는 소자의 이름을 입력하거나 〈F9〉를 눌러 선택번호로 소자를 선택하면 된다.

```
Type in a Unit designation
```

이란 메시지가 나타나면 소자의 이름을 간략하게 기입한다. (예를 들면 74-LS04 를 LS04로)한개의 소자를 복수개 사용할 경우에는, 이름을 한 자로 기입하면 자동적으로 번호가 이름 뒤에 차례로 덧붙여 진다. 다시

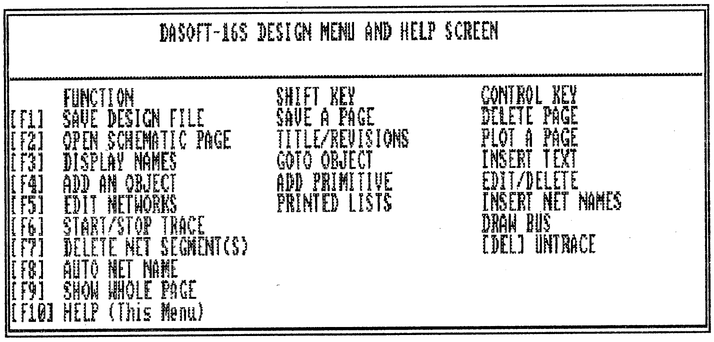
```
Type in the VALUE of the component
```

라는 메시지가 나타난다. 이것은 〈ENTER〉 키를 눌러 무시해도 되지만 사용하고자 할 경우 저항은 저항값을, 콘덴서는 용량을 기입한다.

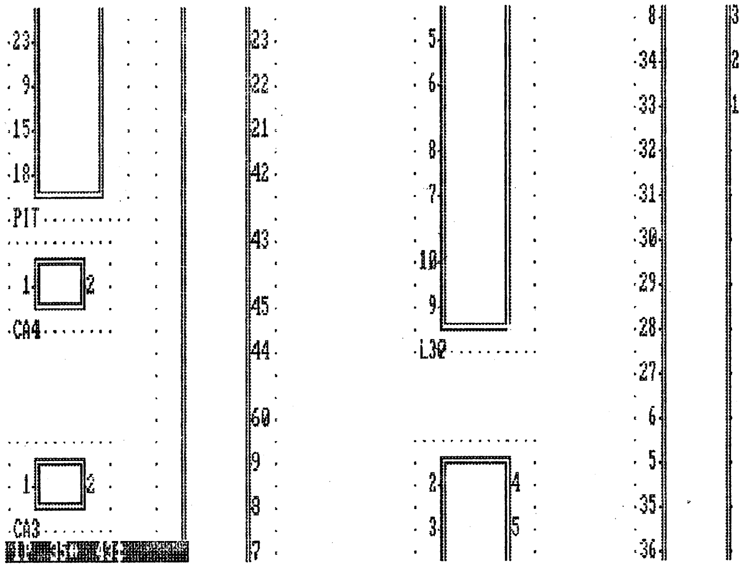
```
Enter ROW of LL CORNER.
[F9]=Cursor Position
```

이란 메시지가 나타나면 〈F9〉를 누르면 되며 커서를 원하는 위치에 옮기지 못했으면 〈F9〉키 대신에 좌표를 지정

〔화면 4〕 다소프트 디자인 메뉴



〔화면 5〕 실제 사용되는 소자를 원하는 갯수만큼 배치한다



해 주어도 된다. 앞에서와 같은 방식으로 실제 사용되는 모든 소자를 원하는 갯수만큼 배치한다(화면 5 참조).

다음은 〈F5〉키를 이용해 네트(NET)를 만드는 작업이다. 네트란 소자와 소자를 연결할 부분들을 무리지어 놓는 것을 뜻하는데 〈F5〉를 누르면 네트이름을 지정받는 화면이 나온다. 새로운 네트 이름일 때는 첫 글자가 알파벳으로 시작하여야 하며 이름은 어떤 부분을 연결할 것인지 알아 보기 쉽도록 정하는 것이 좋다.

여기서는 88번의 VCC를 예로 들어 네트를 연결하는 방법을 살펴 보기로 하자. **화면 6**이 나타나면 →키를 이용해 **화면 7**의 형태로 간다. 연결할 핀을 갖고 있는 소자의 오른쪽 끝에 커서를 이동시킨 다음 〈F9〉를 누르고 다시 해당 핀 번호에 커서를 옮겨서 〈Ins〉키를 누르면 네트에 포함되게 된다. 이처럼 한 네트에 필요한 모든 부분을 구성해 놓은 후 〈F5〉로 나오면 하나의 네트가 만들어 지는 것이다. 이와 같은 방법으로 모든 네트를 만든 후 〈F1〉키를 눌러 저장한다. 그런데 네트를 설정하는 또다른 방법으로 〈F8〉키를 이용하는 방법이 있는데 그 방법은 사용자가 알아보기 어려운 경우가 있기 때문에 가급적 사용을 피하는 것이 좋다.

〈F9〉키는 전체 페이지를 한 화면으로 보여 주면서 아울러 페이지상에서 원하는 소자를 찾아 커서를 손쉽게 옮길 수 있다.

DB.EXE의 실행

D.EXE에서 능동부소자와 수동소자와 Footprint와 스키매틱을 완료하고 저장했으면 〈Ins〉키를 눌러 DOS 상으로 빠져 나온다. 여기서 DB를 입력해 메인 메뉴로 들어 간다. 그 메뉴를 **화면 8**에 보였다.

먼저 'C'를 눌러 사용하고자 하는 라이브러리 디스켓이 들어 있는 드라이브를 지정한다. 아마 대개는 B드라이브가 될 것이다. 한가지 주의할 점은 다소프트는 매우 많은 화일들을 만들기 때문에 저장하려는 디스켓에 충분히 빈 공간이 있어야 한다. 만약 그렇지 못할 경우는 저장하면서 전에 만든 화일과 겹쳐버리는 경우도 있다. (필자 자신도 이런 문제로 하드디스크를 망친 일도 있으며 지금은 새 화일을 만들 때마다 새 디스켓을 사용한다.)

'C'를 눌러 드라이브를 바꾸었으면 'B'를 눌러 보드 에디터모드로 들어간다. 그러면

```
Enter the name of an Outline
file or press[F9]
```

와 같이 새로 화일을 작성할 것인지 물어 본다. 우선 새로 화일을 만들 경우를 보자. 먼저 화일이름을 입력해 넣는다. (예로 JJS) 그러면 새로운 화

〔화면 6〕 VCC 선택 화면(1)

```
VCC
------------------------------------------------------------------------------------
UNIT TYPE  COMPONENT   P#  NAME  LTR(S) |UNIT  NAME        PGE ROW COL
S1         SLOT-16BIT  34  +5V          |S ╔══════════════════════════╗16
S1         SLOT-16BIT  60  5V           |L ║[↕] to Scan List.         ║44
L139       74S139      16  VCC          |L ║[Del] Delete a segment    ║44
L30        74LS30      14  VCC          |L ║[→] Show Parts List       ║44
L40   INV  7404        14  VCC          |P ║[F5] Exit Net Editor      ║68
PPI        8255A       26  VCC          |C ║[F4] Edit Net Name        ║89
L30        74LS30       1  A            |C ╚══════════════════════════╝ 0
CA         CAPA         1  VCC          |PCA   P CAPA       1   12   0
PCA        P CAPA       1  +            |CA2   CAPA         1   19   0
CA2        CAPA         1  VCC          |CA3   CAPA         1   27   0
CA3        CAPA         1  VCC          |CA4   CAPA         1   36   0
CA4        CAPA         1  VCC          |S60   SLOT-60      1   22 106
OSC        CLOCK-OSC   14  VDD          |PPI2  8255A        0    0   0
PPI1       8255A       26  VCC          |OSC   CLOCK-OSC    1   56  67
PIT        8253        24  VCC          |PPI1  8255A        1   22 123
S60        SLOT-60      3  3            |PIT   8253         1   41   0
S60        SLOT-60      4  4            |CA5   P CAPA       1   73  18
CA5        P CAPA       1  +            |CA6   P CAPA       1   79  18
CA6        P CAPA       1  +            |CA7   P CAPA       1   84  18
CA7        P CAPA       1  +            |CA8   P CAPA       1   84  32
```

〔화면 7〕 VCC 선택 화면(2)

```
VCC
------------------------------------------------------------------------------------
UNIT TYPE  COMPONENT   P#  NAME  LTR(S) |UNIT  NAME        PGE ROW COL
╔═════════════════════════════╗4  +5V   |S1    SLOT-16BIT   1    6  16
║[↕] to Scan List.            ║0  5V    |L139  74S139       1    6  44
║[F9] List Pins in Object.    ║6  VCC   |L30   74LS30       1   35  44
║[Ins] Add Pin to Net.        ║4  VCC   |L40   7404         1   65  44
║[←] Show Net List.           ║4  VCC   |PPI   8255A        1    4  68
║[F5] Exit Net Editor.        ║6  VCC   |C1    34P CONNEC   1   22  89
╚═════════════════════════════╝1  A     |CA    CAPA         1    4   0
CA         CAPA         1  VCC          |PCA   P CAPA       1   12   0
PCA        P CAPA       1  +            |CA2   CAPA         1   19   0
CA2        CAPA         1  VCC          |CA3   CAPA         1   27   0
CA3        CAPA         1  VCC          |CA4   CAPA         1   36   0
CA4        CAPA         1  VCC          |S60   SLOT-60      1   22 106
OSC        CLOCK-OSC   14  VDD          |PPI2  8255A        0    0   0
PPI1       8255A       26  VCC          |OSC   CLOCK-OSC    1   56  67
PIT        8253        24  VCC          |PPI1  8255A        1   22 123
S60        SLOT-60      3  3            |PIT   8253         1   41   0
S60        SLOT-60      4  4            |CA5   P CAPA       1   73  18
CA5        P CAPA       1  +            |CA6   P CAPA       1   79  18
CA6        P CAPA       1  +            |CA7   P CAPA       1   84  18
CA7        P CAPA       1  +            |CA8   P CAPA       1   84  32
```

일을 만들 것인지 다음과 같이

That is a New file CREATE? (Y/N)

물어 온다. 이때 'Y'를 입력하고 그 다음에는 자기가 그릴 보드의 폭을 써넣는다. 그후 보드의 높이를 써 넣는다. 화면 9는 폭과 넓이를 지정하고 난 후의 화면이고 여기서 쓰이는 명령어들은 표 1에서 정리하였다. 이 보드에디터에서 작성한 작업은 다음 단계에서 설명할 레이아웃에서 다시 작성할 수 있으며 오히려 아웃 라인(Outline)을 잘못 설정할 경우 레이아웃상에서 수정하기가 귀찮으므로 그대로 선 하나 정도만 그리고 나가기로 하자. (나갈 때는 〈F1〉으로 저장하고 나가든가 아니면 〈ESC〉로 빠져 나가든가 하면 된다.) 주의할 것은 이 아웃라인 모드에서 아무 작업도 하지 않고 나가면 에러가 발생한다는 것이다.

다음은 수정할 경우를 보자. 수정할 경우는 〈F9〉키를 눌러 수정하고자 하는 화일의 번호를 선택하고 〈ENTER〉키를 치면 아웃라인 모드로 들어 간다. 그 이하는 앞에서 설명한 바와 같다.

이제 다소프트의 핵심 부분인 레이아웃을 살펴 보자. 먼저 'L'을 눌러 레이아웃을 선택한다. 그러면

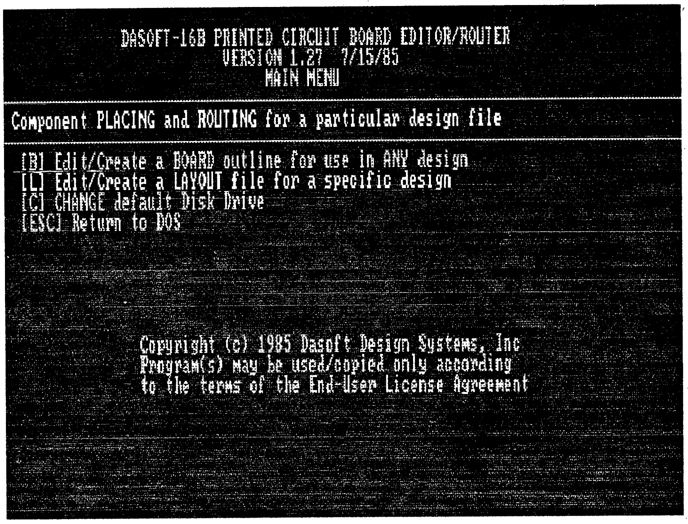
Disk space Available=28K ABORT Y/N ?

라는 메시지가 출력된다. 이것은 디스크의 남은 용량을 표시해 주고 작업을 계속할 것인가를 물어 보는 것이다. 앞에서도 언급했지만 저장할 충분한 용량이 남아 있어야 한다는 것을 명심하자. 충분한 용량이 남아 있으면 'L'을 누른다. 그러면

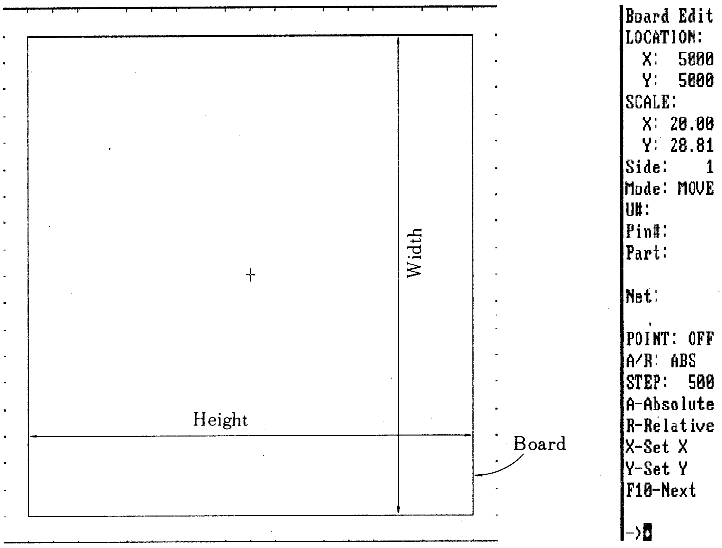
Enter the name of a LAYOUT file or press[F9]

라는 메시지가 출력되며 새로 화일을 만들 것인지 이미 작성된 화일을 수정할 것인지 물어 온다. 수정을 할 경우

〔화면 8〕 DB.EXE의 메인 메뉴



〔화면 9〕 보드의 폭과 넓이 지정



에는 이미 여러번 설명했으므로 이번에는 새로 작성할 경우에 대해서 살펴보기로 하겠다. 먼저 화일이름을 입력한다. 그러면

That is a NEW file CREATE ? Y/N

라는 메시지가 출력되고 새로 작성하는 화일이므로 'Y'를 누르면 된다. 그 다음엔

Enter the name of DESIGN or press[F9]

라는 메시지가 출력되고 독자 여러분이 디자인한 화일이름을 알고 있을 경우는 직접 입력해 넣으며 모를 경우에는 〈F9〉로 리스트를 출력하여 번호를 선택하면 된다. 다음은

Enter the name of a OUTLINE file or press[F9]

〔표 1〕 보드 편집용 명령어

키	기 능
F1	작업 내용을 저장
F2	뒷면을 보여준다
F3	선을 그린다
F4	구멍을 뚫는다
F5	F3기능과 유사
F6	축소, 확대
F7	화면을 다시 생성
F8	사용하지 않음
F9	사용하지 않음
F10	다음 메뉴를 보여준다
A	Absolute Move
R	Relative Move
X	X좌표 지정
Y	Y좌표 지정

라는 메시지가 출력되는데 앞에서 설명한 것처럼 아웃라인 화일을 입력해 넣으면 된다. 그러면 잠시 후에 화면이 나타나 작업을 할 수 있는 공간이 나온다.

지금까지의 과정은 앞으로의 작업을 위해서 D(Dasoft)에서 Footprint와 스키매틱을 한 것이다. 따라서 이 부분이 가장 중요하다고 볼 수 있다. 레터링을 해 보았던 독자라면 쉽게 적응할 수 있을 것이다.

먼저 〈F10〉을 누른다. 그럼 화면 10이 출력될 것이다. 이것이 이 모드에서 사용되는 모든 명령어들이다. 이 명령어들을 설명하면서 실제 작성한 예를 들겠다. 〈F4〉를 누르면 오른쪽 상단에 이 표시되고 여기에 찍고자하는 부품의 유니트(Unit) 이름을 입력하고 〈ENTER〉를 친다. 그 다음에는 부품을 입력할 위치를 표시하는데 커서를 부품이 놓일 위치에 놓고 그 위치의 X좌표를 쓰고 〈ENTER〉를 누르고 Y좌표를 입력하고 〈ENTER〉를 누른다. 그런 후에는 부품이 표시될 방향을 1~4번까지 선택하는데 이것은 그림 2를 참조하기 바란다. 몇번이고 연습해서 자유롭게 쓸 수 있도록 하자. 간혹 방향까지 다 선택하고 난 후에도 다음 부품이 표시되지 않을 경우가 있는데 그때는 'INTER'나 'GRID'로 표시되지 않은 이유가 오른쪽 하단에 표시된다.

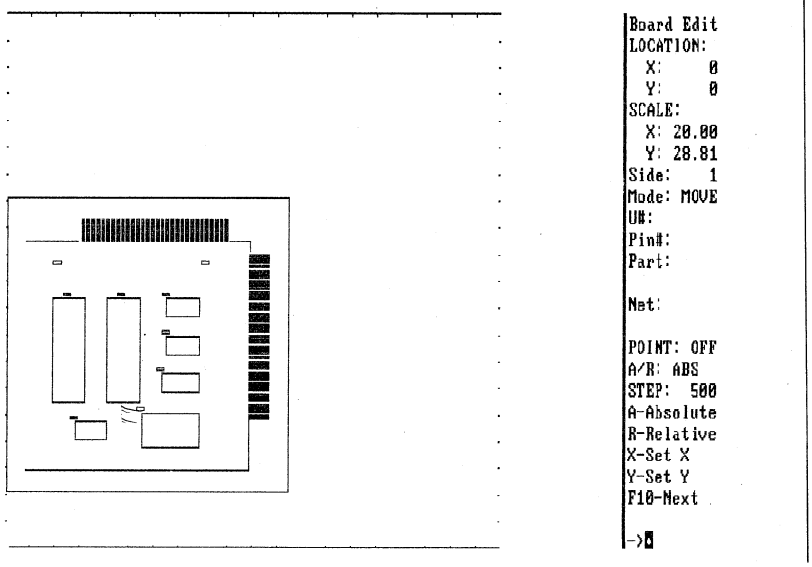
'INTER'란 다른 부품과 겹쳤다는 뜻이고 'GRID'는 부품이 선위에 있다는 뜻이다. 이럴 경우에는 방향을 바꾸거나 위치를 옮겨서 표시하도록 하자. 화면에 나타낼때 기준이 되는 것은 항상 1번이며 옮길 때는 옮길 위치에 커서를 이동시켜 〈F4〉를 눌러 표시하면 되고 지울 때는 〈F4〉를 누르고 유니트 이름을 쓴후 〈DEL〉키를 누르면 된다. 그리고 표시한 위치나 전체 부품의 배치를 보기 위해서는 〈Ctrl〉-〈F1〉을 누른다. 그 결과를 화면 11에 나타내었다. 이 화면을 자주 보면서 전체 부품의 배치를 알맞게 하도록 한다. 빠져 나갈 때는 〈ESC〉키를 누르면 된다. 부품의

〔화면 10〕 레이아웃 화면에서의 〈F10〉 선택

```
F1    = SAVE                      C-F1  = EDIT BOARD FEATURES      Layout Ed
F2    = TOGGLE SIDE               C-F2  = PLOT BOARD               MILS:
F3    = GOTO OBJECT               C-F3  = CENTER SCREEN              X:    0
F4    = ADD/DELETE/MOVE OBJECT    C-F4  = PRINTED LISTS              Y:    0
F5    = AUTO ROUTER               C-F5  = CHANNEL ROUTER           GRIDS
F6    = ADD NORMAL TRACE          C-F6  = ADD FAT TRACE              X:    0
F7    = SHOW UNROUTE ON SCREEN    C-F7  = DELETE NET SEG             Y:    0
F8    = TURN OFF UNROUTE          C-F8  = TAG TO REROUTE           Side:   1
F9    = LIST                      C-F9  = REROUTE FROM TAG         Mode:
F10   = DISPLAY HELP              C-F10 = REROUTE FAT FROM TAG     U#:
                                                                   Pin#:
V     = VIEW (WINDOW) OTHER SIDE  G     = ADD/DELETE GRID          Part:
?     = FLASH PINS ON NET         /     = CLEAR FLASH
INS   = ADD/DELETE VIA            DEL   = DELETE TRACE             Net:

                                                                   ->
```

〔화면 11〕 전체 부품의 배치



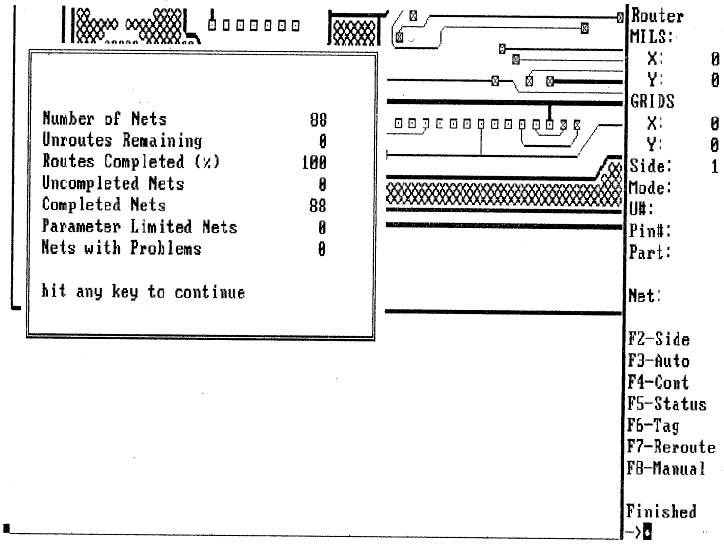
배치가 끝났으면 부품 상호간의 선을 배선해 보도록 하자. 〈F5〉를 누르면 F5에 대한 메뉴가 나온다. 여기서 다시 〈F3〉을 누르면 컴퓨터가 자동으로 배선을 해 준다. 배선이 끝나면 〈F5〉를 눌러 보자. 그러면 화면 12와 같이 전체 네트 수, 연결안된 선의 수, 몇 %가 완성되었는가 등이 나타난다. 컴퓨터가 배선해 주지 못한 선들은 직접 배선해 주어야 한다. 이제 직접 배선하는 방법을 알아 보자.

먼저 각 부품의 패드 위를 커서로 지나가 보자. 그러면 그 패드에 관한 정보가 오른쪽에 나타날 것이다. 그 중에서 네트 이름이 나타나는 패드 위에서 '?'를 입력해 보자. 그러면 그 핀과 연결되어야 할 핀들이 반짝거릴 것이다. 〈F6〉을 누르면 모드가 "ADD"로 바뀔 것이다. 숫자판의 1~9번 사이의 키를 눌러 전후좌우 대각선으로 움직여 반짝이는 두 점 사이를 연결한다. 잘못 그린 경우는 'DEL'을 눌러 선을 따라가면서 지우면 된다.

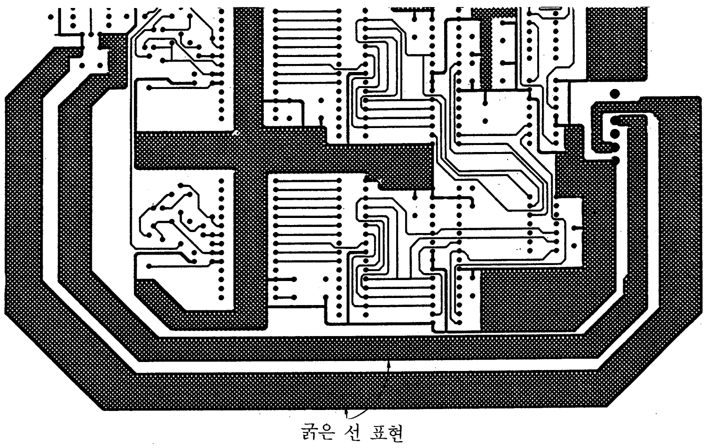
전원선같이 많은 전류를 흘려야 할 선들은 〈Ctrl〉-〈F6〉을 누르면 굵은 선으로 배선할 수 있다. 또한 화면 13과 같이 굵은선으로 만들 수도 있다. 그리드를 사용하는 법은 'G'를 누르고 커서를 움직이면 된다. 지우고자 할 때는 'DEL'키를 누르고 'G'를 누르고 커서를 움직여 지우면 된다. 이런 식으로 반짝거리는 점들을 모두 연결했으면 '/'키를 눌러서 반짝거리는 명령을 해제시킨다.

연결해야 하는 선들을 찾는 명령에는 또 한 가지가 있다. 그것은 〈F7〉이다. 네트 이름이 있는 핀 위에서 〈F7〉을 누르면 연결되야 하는 핀이 가상적인 굵은 선으로 연결되어 표시된다. 다른 곳에서 〈F7〉을 누르면 'NET?/[F9]'라는 메시지가 나오고 〈F9〉를 누르면 전체 네트의 리스트가 나온다. 여기서 반전된 부분의 네트가 연결안된 네트이고, 보통 글자체의 네트는 이미 연결된 네트이다. 연결안된 네트중의

〔화면 12〕 배선에서의 전체 네트 수, 연결 안된 선의 수, 완성된 정도를 보여준다

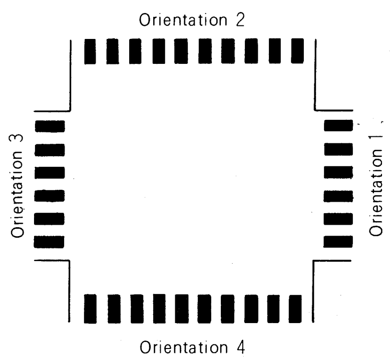


〔화면 13〕 굵은 선의 표현



〔그림 2〕 부품을 그리는 방향 선택

Orientation	1	2	3	4
pin1~pin2	1 ~ 2	1~2	2 ~ 1	2~1



네트이름을 입력하면 화면이 바뀌면서 연결되야 할 두 곳을 가상적인 굵은 선으로 연결해 줄 것이다. 만약 "off board"가 출력되면 그 부품이 표시되지 않은 것이므로 〈F4〉로 찍어야 한다. 또 배선을 다시 해주고 싶을 땐 꼭 일일이 지우지 않고도 배선해 주는 방법이 있는데 얇은 선일 경우는 〈F6〉+〈Ctrl〉-〈F8〉-〈F9〉이고 굵은 선일 경우는 〈Ctrl〉-〈F6〉-〈F8〉-〈F9〉이다. 실제로 해보면 쉽게 이해가 될 것이다.

기판이 양면일 경우에는 앞뒷면을 다 보아야 할 필요가 있다. 이때는 〈F2〉를 누르면 사용하고 있는 화면의 뒷면을 볼 수 있다. 이와 비슷한 기능으로 커서 주변의 뒷면을 볼 수 있는 기능도 있는데 그것이 'V'이다. 또 배선을 하다 보면 뚫어서 배선해야 하는 경우도 있는데 이 뚫는 구멍을 바이어(Via)라고 한다.

구멍을 뚫는 방법은 다음과 같다.

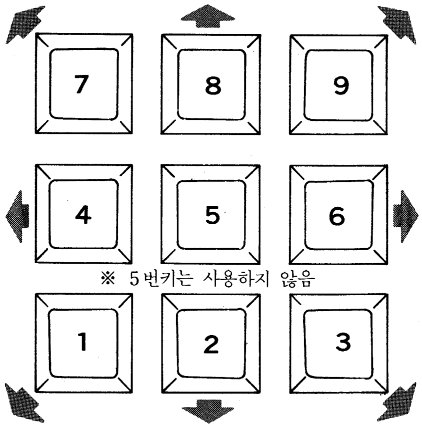
〈Ins〉키를 누르면 'Via? Y/N'이라는 메시지가 나오므로 구멍을 뚫고 싶으면 'Y'를 입력한다. 만약 잘못 뚫었거나 지우고 싶다면 'DEL' 키를 누르고 〈Ins〉키를 누르면 된다. 이때

```
Fix up metal on both sides of
board !
```

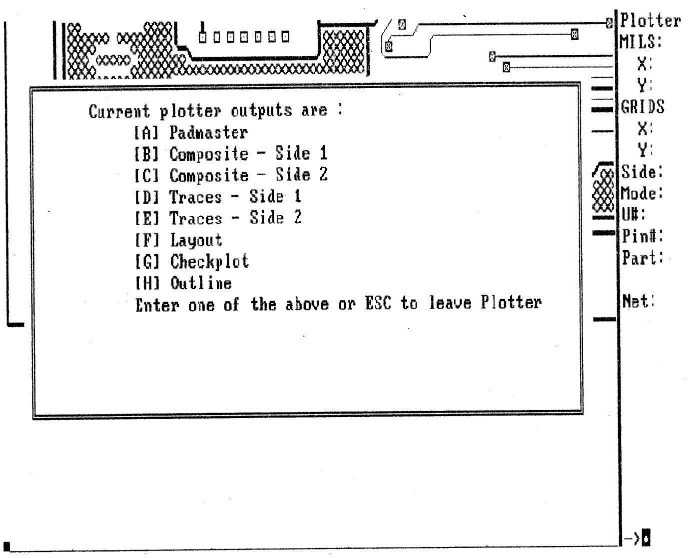
라는 메시지가 출력되는데 여기서 〈ENTER〉 키를 누르면 메시지가 사라지면서 바이어가 없어진다. 다음에는 커서를 이동하는 방법에 대해 자세히 알아 보자. 숫자판에 있는 숫자를 〈NumLock〉키를 누르지 않은 상태에서 커서키로 사용하면 된다(그림 3 참조).

커서를 한꺼번에 많이 움직이고 싶은 경우에는 좌우 방향으로는 〈Ctrl〉←, →키를 누르면 되고 그외의 방향은 움직이고 싶은 거리만큼 숫자를 써주고 움직이고자 하는 방향의 키를 눌러주면 된다. (예를 들면 100↑ : 윗방향으

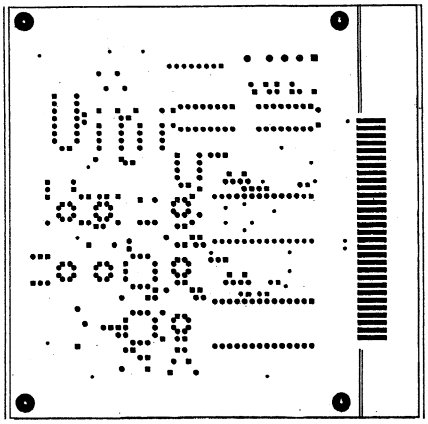
〔그림 3〕 숫자키 패드의 키방향



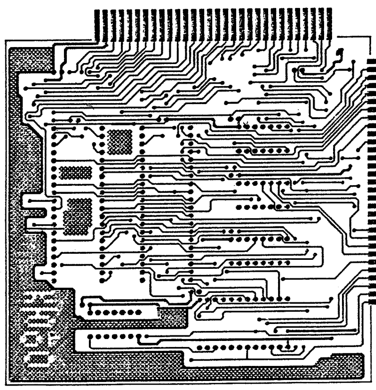
〔화면 14〕 플로터 출력 선택 화면



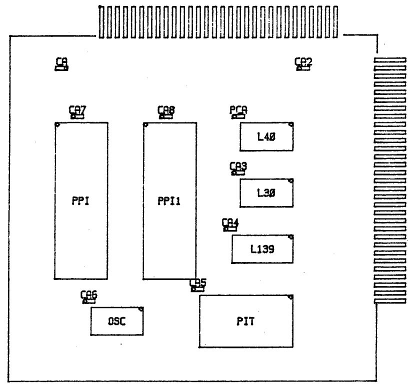
〔화면 15〕 패드 출력(A선택)



〔화면 16〕 모든 작업내용 출력(B선택)



〔화면 17〕 부품배치와 부품명 출력(F선택)



로 100만큼 이동) 또 원하는 부품으로 바로 가고자할 때는 〈F3〉을 누르고 유니트이름을 써주면 바로 그 부품으로 갈 수 있다. 또 〈Ctrl〉-〈F1〉상에서 커서를 부품 근방으로 위치시키고 〈Esc〉를 누르면 그 부품근방의 화면이 나타나게 된다. 또 〈Ctrl〉-〈F3〉은 커서를 그 화면의 중앙으로 옮길 경우에 쓰인다.

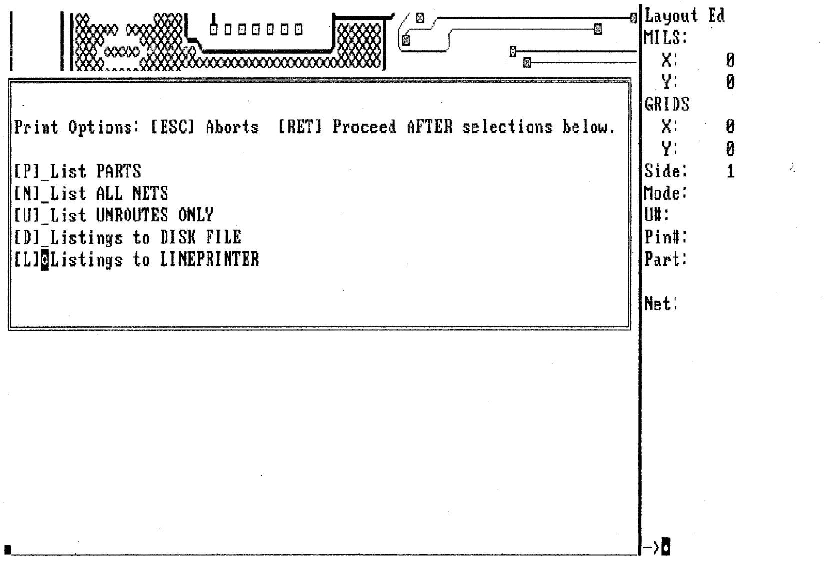
이로써 작업상에 필요한 명령어는 모두 설명한 셈이다. 이제 저장하는데 필요한 명령어와 플로터에 관해 알아보자. 먼저 저장하기 위해서든 〈F1〉키를 누른다. 그러면

```
SAVE the layout ? Y/N
```

이라는 메시지가 나오고 'Y'를 누르면 저장된다. 또 DOS상으로 빠져 나오기 위하여 〈ESC〉를 누르는 경우가 있는데 저장할 경우에 〈ESC〉를 잘못 눌러 그동안 애써 만든 화일이 몽땅 날라가 버리는 일이 없도록 주의해야 할 것이다. 다음은 플로터에 관한 명령들이다.

작업이 완성되고 플로터로 출력을 하고자 하면 〈Ctrl〉-〈F2〉를 누른다. 그러면 화면 14와 같은 메뉴가 나타난다. 여기서 'A'를 선택하면 패드만이 그려진다(화면 15 참조). 'B'를 선택하면 작업한 모든 것이 찍힌다(화면 16 참조). 'C'

〔화면 18〕 프린터 출력 선택 화면



도 마찬가지이다.

'D'는 선과 그리드만을 그리고 'F'는 부품의 배치와 부품명을 그린다(화면 17 참조). 'G'는 플로터의 상태를 시험하는 것이다.

다음은 네트가 과연 올바르게 연결되어 있는지 쉽게 확인하기 위하여 프린터로 출력하는 명령이다. 〈Ctrl〉-〈F4〉를 누르면 화면 18과 같은 메뉴가 나오는데 여기서 'P'는 사용된 부품이 어디 있는지 확인하는 것이며 'N'은 연결된 전체 네트를 확인하는 것이다. 그밖의 'U', 'D', 'L' 등은 사용할 경우가 거의 없는 것으로 실제 사용해 보면 쉽게 알 수 있을 것이다.

이상과 같이 간단하게 다소프트의 사용법에 대해서 알아보았는데 지면관계상 전부는 언급하지 못하고 말았다. 세세한 사항은 매뉴얼 등을 참고하기 바라며 각자의 실무에 많은 도움이 되길 바란다.

보내는 사람

□□□-□□□

우편요금
수취인후납부담
발송유효기간
1989. 2. 1~1991. 12. 31
서울 체신청
승인119호

월간 퍼스널컴퓨터 하이테크

서울특별시 용산구 갈월동 17

퍼스널컴퓨터 편집부앞

140-150

월간 퍼스널컴퓨터의 자매기관

애독자페이지	관심있는 분야		게재를 원하는 기사	
	□ 컴퓨터에 관한 알고 싶은 질문			
	□ 본지에 바라는 의견			
정기구독신청서	구독신청방법	□ 은행지로 □ 우체국 대체구좌 □ 소액환 □ 기타		
	구독기간	년 월호부터 년 월호까지	구독신청일자	
	□ 받으실 분의 이름과 주소(우편번호) ☎ (—)			

※ 이곳에 풀칠을 하여 뒷면 수신란이 나오게 접어서 우체통에 넣어 주세요(우표부착 불필요).

애독자 설문지

책의 편집에 반영하고자 하오니 성의껏 적어서 보내주시기 바랍니다.

<table>
<tr><td rowspan="2">작성자</td><td>이름</td><td></td><td>나이</td><td></td><td>직장</td><td></td><td>성별</td><td></td></tr>
<tr><td>주소</td><td colspan="7">☎ (　　－　　)</td></tr>
<tr><td rowspan="5">사용기종</td><td>8비트</td><td colspan="2"></td><td>16비트</td><td colspan="2"></td><td>32비트</td><td></td></tr>
<tr><td>주변기기</td><td colspan="7">□ 모니터(컬러, 단색) □ 드라이브(HDD, FDD) □ 프린터(9핀, 24핀, 레이저) □ 인터페이스카드(　　)</td></tr>
<tr><td>사용용도</td><td colspan="5">□ 취미 □ 업무 □ 교육 □ 학습 □ 기타</td><td>사용경력</td><td></td></tr>
<tr><td>사용언어</td><td colspan="7">□ 베이직 □ C □ 파스칼 □ 기계어 □ 기타</td></tr>
<tr><td>사용분야</td><td colspan="3"></td><td>사용 소프트웨어</td><td colspan="3"></td></tr>
<tr><td rowspan="5">책 구입 방법</td><td>구입방법</td><td colspan="7">□ 정기구독 □ 서점구입(　　서점) □ 증정 □ 기타</td></tr>
<tr><td>구입동기</td><td colspan="7">□ 정기구독 □ 필요한 내용이 있어서 □ 주위에서 권해서 □ 기타</td></tr>
<tr><td>본지외 구독잡지</td><td colspan="3">□ 없다 □ 있다(잡지명 :　　)</td><td>원하는 발행일자</td><td colspan="3"></td></tr>
<tr><td>애로사항</td><td colspan="7">□ 가격이 비싸다 □ 종이 질이 나쁘다 □ 인쇄상태가 나쁘다 □ 기타</td></tr>
<tr><td colspan="8">□ 본지 구독을 추천하고 싶은 사람이 있다면(이름과 주소)
☎ (　　－　　)</td></tr>
<tr><td rowspan="2">이번달에 좋았던 기사(2가지)</td><td>제목</td><td colspan="3"></td><td>제목</td><td colspan="3"></td></tr>
<tr><td colspan="4">□ 이유</td><td colspan="4">□ 이유</td></tr>
<tr><td rowspan="2">이번달에 싫었던 기사(2가지)</td><td>제목</td><td colspan="3"></td><td>제목</td><td colspan="3"></td></tr>
<tr><td colspan="4">□ 이유</td><td colspan="4">□ 이유</td></tr>
</table>

자르는 선

월간 퍼스널컴퓨터의 정기구독자에게는 아래와 같은 특전이 마련되어 있습니다.

—정기구독자는 본사가 보유한 각종 정보 및 자료를 무료로 배부받으실 수 있읍니다.
—본지에 게재되는 각종 프로그램을 디스켓에 담아 무료로 보내드립니다.
—본지 및 본지의 자매지(월간 전자과학) 주최로 개최되는 세미나 및 각종 행사에 무료로 참석하실 수 있읍니다.
—본지에서 행하는 각종 부대사업에 할인혜택이 주어집니다.
—구독기간중 책 가격이 인상되더라도 종전가격으로 구독이 가능합니다.

●구독신청은

—시중은행에서 지로용지를 이용하시거나 우체국에서 대체구좌 또는 소액환으로 송금하시면 됩니다.
은행 지로번호 : 3000667
우체국 대체구좌번호 : 010256—31—0526772

●정기구독료는

—1 년 구독시 30,000원
—6개월 구독시 15,000원

●구독신청시 주의사항은

-구독기간과 주소(우편번호), 전화번호와 받으실 분의 성명을 정확히 기입하시기 바랍니다.
—배달사고나 기타 변동사항은 본지 영업부로 즉시 연락하시기 바랍니다. 신고전의 분실사고는 책임지지 않습니다.

●정기구독 문의처는

—서울시 용산구 갈월동 17 ㊍ 140-150
월간 퍼스널컴퓨터 영업부
전 화 : (02)754—1044(대표)
FAX : 757—1049

▨ 전국총판 ▨

●서울●	종로, 중구, 마포, 용산	명 문 사	774—6314
	중랑, 성동, 동대문	홍 익 도 서	454—1810
	노원, 성북, 도봉	진 원 사	981—4884
	강남, 강동, 송파	광 일 사	415—7330
	관악, 동작	관 악 사	889—4504
	서대문, 은평	형 제 사	304—9462
	영등포, 구로	영 도 사	613—5381
	강서, 양천	연 우 사	865—8239
		진 명 서 점	272—5871
		관 음 서 적	266—7792
●부산●		종합기술서적	462—2431
		영광도서전시관	805—9500
		북 부 총 판	805—9677
		남 부 총 판	756—2727
●대구●		청 운 서 림	23—7335
		한 일 서 적	23—7101
		학 원 사	22—0336
●인천●		동 양 서 점	73—3384
●광주●		삼 복 서 점	22—0258
●경기●	부천	대 경 서 점	62—3672
	안양, 안산	사 랑 글 방	57—1916
	수원	동 아 서 점	5—6436
	성남	영 광 서 점	2—6532
●강원●	원주	원주종로서적	42—2943
	삼척	영 동 사	72—6060
	강릉	삼 일 사	2—2314
	춘천	대 진 서 점	54—5148
●충북●	충주	문 학 사	2—2814
	청주	화 문 당	55—0503
●충남●	천안	동방서적공사	63—3314
	대전	동 양 서 점	22—1538
●경북●	포항	경 북 서 점	2—2717
	구미	중 앙 서 점	52—7915
●경남●	진주	동 아 서 점	42—3768
	진해	학 애 서 점	2—2523
	울산	우 정 서 점	43—6003
	마산	기 술 서 적	56—1417
●전북●	전주	홍 지 서 점	6—5311
	군산	동 원 서 점	42—3776
●전남●	목포	문 화 당	5—3086
	여수	대 양 서 점	62—2111
●제주●		광 양 서 적	53—5077

모니터

사람들이 입을 통하여 자신의 의사를 표현하는 것처럼 컴퓨터는 모니터를 통하여 자신의 상태를 표현하고 있다. 이와같이 모니터는 컴퓨터에서 처리한 결과를 화면으로 출력시켜 주고, 입력되는 데이터를 확인시켜 주기도 한다. 또한 이 모니터는 사용자에 따라 여러가지 모습으로 변신하기도 한다. 때로는 제도판과 같은 역할을 하기도 하고 캔버스가 되기도 하는 다양한 기능을 제공하기도 한다. 하지만 이러한 기능은 모니터 혼자의 힘으로 할 수 있는 기능이 아니고 메인시스템의 도움을 받아야 만이 가능하다.

●모니터의 선택

모니터의 선택에 앞서 우선적으로 고려해야 할 것은 어떤 용도로 사용할 것인지에 대한 선택이 우선적으로 고려되어야 한다는 것이다. 왜냐하면 모니터는 위에서 이야기한 것처럼 여러가지의 모습을 하고 있기 때문이다. 그 여러가지의 모습중에서 자기가 필요한 모양을 선택하여야 한다. 필요한 모습을 선택하기 위해서는 어떠한 모습들이 있는가를 살펴보고 결정하는 것이 순서일 것이다.

●모니터의 종류

모니터의 종류는 TTL모노크롬 모니터와 CGA용 컬러모니터, EGA용 컬러모니터, VGA용 컬러모니터, 멀티싱크(Muiti-sync) 모니터 등이 있다.

모니터의 종류을 구분하는데는 여러가지의 방법이 있는데 간략하게 소개하면 컬러에 의한 방법과 입력신호에 의한 방법등이 있다. 컬라에 의한 분류방법은 컬러와 단색(보통은 녹색)으로 구분하는 것이다. 입력신호에 의한 방법은 TTL모니터와 아날로그모니터로 구별할 수 있다. 이처럼 구분하는 방법이 다양한 것은 모니터가 능동적인 기능을 갖고 있지 못하고 비디오 보드의 지원에 따른 수동적인 역할 때문으로 풀이된다.

TTL모노크롬 모니터에서 사용가능한 보드는 MDA(Monochrome Display Adapter) 보드와 허클리스 보드 그리고 EGA 보드가 사용가능하다. MDA 보드 사용시는 텍스트모드에서는 80×25행의 문자를 표현할 수 있으며 허클리스 보드에서는 그래픽모드에서 720×350의 해상도를 지원한다. EGA 보드도 모노크롬모니터에서 사용이 가능하지만 실질적으로 모노크롬 모니터에서 EGA보드를 사용하는 경우는 거의 없는 상황이고 또한 사용한다 하더라도 완벽하게 동작되지는 않는다.

CGA용 모니터는 CGA(Color Graphic Adapter)보드와 EGA 보드에서 사용이 가능하다. CGA 보드상에서는 텍스트모드, 중간 해상도의 그래픽 모드, 고해상도 그래픽모드 등이 있는데 고해상도 그래픽 모드상에서 640×200의 해상도를 표현할 수 있고 한 화면상에서 4가지의 컬러를 지정할 수 있다. 그리고 배경컬러는 16컬러중에 1가지를 선택할 수 있다. EGA보드상에서도 CGA 보드와 비슷한데 한 화면에 16가지의 컬러를 선택하여 사용할 수 있는 것이 특징이다.

EGA용 모니터에 사용가능한 비디오 카드는 CGA보드와 EGA 보드이다. CGA보드를 사용할 경우는 16컬러중 4가지의 컬러를 지정하여 사용할 수 있고 배경 색상으로는 1가지를 선택할 수 있다. EGA보드 사용시는 16컬러의 지정과, 640×350의 해상도를 표현한다.

VGA용 모니터는 VGA 보드상에서만 작동된다. 그리고 720×480의 해상도를 지원하고 아날로그 RGB신호를 입력받아 256가지의 컬러 사용이 가능하다.

이렇듯 모니터의 종류에 따라 비디오 보드가 달라지는데, 이것은 비디오 보드가 바뀌면 모니터를 바꾸어 주어야 하는 불편함이 존재하고 있다. 따라서 비디오 보드의 발표때 마다 새로운 모니터의 개발이 필수적으로 따라주어야 했다. 하지만 이러한 불편함을 불식시키기 위한 노력이 NEC에 의하여 추진되어 개발되었다.

즉 거의 모든 비디오 보드를 수용할 수 있는 멀티싱크 모니터의 개발이다. 멀티싱크 모니터의 탄생으로 새로운 보드의 개발에도 의연하게 적응할 수 있게 되었다.

이 멀티싱크 모니터는 TTL과 아날로그 신호를 모두 처리할 수 있다. 그리고 MDA, CGA, EGA, VGA 등을 사용할 수 있다. 그리고 이 모니터는 비디오 보드에서 출력되는 주파수가 15kHz에서 35kHz 사이에 출력하는 모든 보드에서 사용이 가능하다.

모니터는 능동적인 기능을 갖춘 것이 아니고 비디오 보드에서 출력되는 신호를 입력받아 처리하는 수동적인 것이므로 사용자들이 구입할때는 사용용도를 결정하여 용도에 맞는 비디오 보드를 결정하고 그 비디오 보드에 맞는 모니터를 선택하는 것이 현명한 방법이다. 멀티싱크 모니터를 구입한다면 비디오 보드에 관계없이 사용이 가능하다. 비디오 보드는 현재 여러가지의 종류가 있으나 대부분 MDA, CGA, EGA, VGA들의 호환보드들이다.

비디오 보드의 종류

비디오 보드도 모니터와 마찬

가지로 일반적인 평가기준이 픽셀(PIXEL)의 유용치로 표시되며 해상도(Resolution)로 구분할 수 있다. 비디오 보드는 MDA, CGA, EGA, VGA 등이 있다.

MDA(Monochrome Display Adapter)는 81년에 소개된 비디오 보드인데 초록색화면을 갖고 있다. 그리고 텍스트 모드상에서 80×25의 화면을 출력한다. 해상도의 지원은 720×350이고, 한문자를 9×14도트로 표현한다.

CGA(Color Graphic Adapter)는 81년에 소개된 컬러지원 보드인데 빛의 3원색(RGB)을 이용하여 컬러 디스플레이를 지원한다. 또한 점 단위의 그래픽 정보를 표현하고 640×200의 해상도가 지원되고 있다. 한 문자의 출력은 8×8도트로 구성된다.

HGC(Hercules Graphic Card)는 82년에 소개되었고, 모노크롬용 문자출력 어댑터에 그래픽 기능을 첨가한 것으로 720×350의 해상도를 지원할 수 있다.

EGA(Enhanced Graphic Adapter) 는 85년에 소개된 컬러 전용보드이다. 이 보드는 16컬러를 지원할 수 있고 640×350 해상도를 지원한다.

VGA(Video Graphic Adatpter) 는 EGA의 모든 기능을 포함하고 해상도를 더욱 증가시켜 720×400의 해상도를 지원할 수 있으며, 컬러의 기능도 최대 256컬러를 지원할 수 있게 되었다.

고해상도 보드는 해상도를 높인 것으로 그래픽에 적합한 보드이다. 해상도가 1024×780 또는 1280×1024의 해상도를 지원할 수 있는 보드이다.

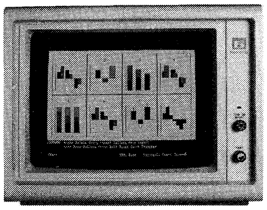
이와같이 모니터와 비디오 보드는 불가분의 관계를 갖고 있다. 이러한 이유에서 모니터의 구입은 비디오 보드를 충분히 고려하여 구입하는 것이 바람직하다고 할 수 있다. 또한 고해상도의 모니터의 대부분이 히다찌, 미쓰비씨, 소니 등의 수입품이 많이 이용되고 있고, 국내에서 생산된 컬러모니터는 대부분이 수출되고 있다.

모니터 사용시 주의사항

모니터는 다른 것과 달리 사용자들이 응급조치를 할 수 없는 것이 대부분이다. 그리고 모니터는 컴퓨터 본체와는 달리 고압의 전류가 흐르고 있기 때문에 상당한 주의를 해야 한다. 그리고 기본적으로 점검해야할 내용은 동작 조건이 사용설명서와 동일한가를 점검하고, 컴퓨터에 사용중인 비디오 보드와 모니터가 일치하는가를 확인하고 시그널 케이블과 연결핀을 확인하는 것이 이외는 사용자가 조치할 수 있는 내용은 없다. 특히 컬러 모니터의 경우는 고압전류가 모니터에서 사용됨으로 사용자가 점검하는 것은 위험성이 따르는 것을 인식하여야 한다.

모노크롬 모니터

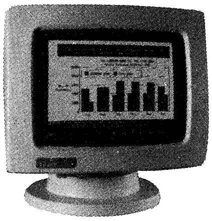
FM1251



- 공급처 : 우정전자
- 전화 : (02) 277-3347
- 크기 : 12인치
- 문자출력수 : 80×25
- 수직주파수 : 50Hz
- 수평주파수 : 18.43kHz
- 입력신호 : TTL Separate
- 가격 : 7만5천원(VAT 별도)

FM1271

- 공급처 : 우정전자
- 전화 : (02) 277-3347
- 크기 : 12인치
- 문자출력수 : 80×25
- 수직주파수 : 50Hz



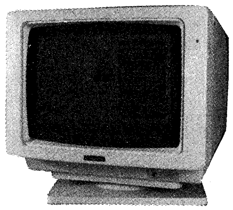
- 수평주파수 : 18.43kHz
- 입력신호 : TTL
- 가격 : 8만원(VAT 별도)

FM1261

- 공급처 : 우정전자
- 전화 : (02) 277-3347
- 크기 : 12인치
- 문자출력수 : 132×24
- 수직주파수 : 60Hz
- 수평주파수 : 15.699kHz
- 입력신호 : Composite
- 가격 : 6만5천원(VAT 별도)

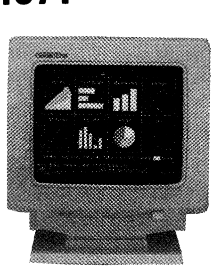
FM1451

- 공급처 : 우정전자



- 전화 : (02) 277-3347
- 크기 : 14인치
- 문자출력수 : 80×25
- 수직주파수 : 50Hz
- 수평주파수 : 18.43kHz
- 입력신호 : TTL Separate
- 가격 : 9만5천원(VAT 별도)

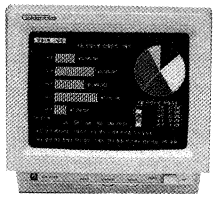
GB-1407T



- 공급처 : 골든벨
- 전화 : (02) 275-2888
- 크기 : 14인치
- 문자출력수 : 80×25

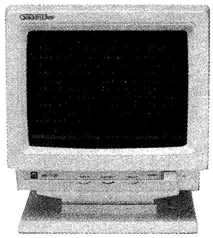
- 수직주파수 : 50Hz
- 수평주파수 : 18.43kHz
- 입력신호 : TTL Separate
- 가격 : 11만원(VAT 별도)

GB-7708A



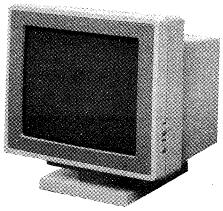
- 공급처 : 골든벨
- 전화 : (02) 275-2888
- 크기 : 12인치
- 문자출력수 : 80×25
- 수직주파수 : 60Hz
- 수평주파수 : 15.75kHz
- 입력신호 : Composite
- 가격 : 8만5천원(VAT 별도)

GB-7708T



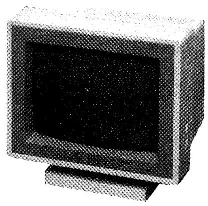
- 공급처 : 골든벨
- 전화 : (02) 275-2888
- 크기 : 12인치
- 문자출력수 : 80×25
- 수직주파수 : 50Hz
- 수평주파수 : 18.43kHz
- 입력신호 : TTL Separate
- 가격 : 8만5천원(VAT 별도)

OM-143TF



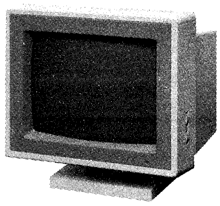
- 공급처 : 오리온 전기
- 전화 : (02) 784-8755
- 크기 : 14인치
- 수직주파수 : 50 / 60Hz
- 수평주파수 : 18 / 20kHz
- 입력신호 : TTL Level
- 기타 : 수출용

OM-123S



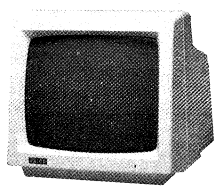
- 공급처 : 오리온 전기
- 전화 : (02) 784-8755
- 크기 : 12인치
- 수직주파수 : 50 / 60 / 70Hz
- 수평주파수 : 31.5kHz
- 입력신호 : 아날로그
- 기타 : 수출용

OM-123T



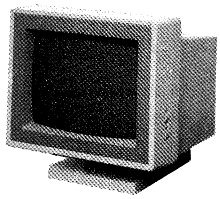
- 공급처 : 오리온 전기
- 전화 : (02) 784-8755
- 크기 : 12인치
- 수직주파수 : 50 / 60Hz
- 수평주파수 : 18 / 20kHz
- 입력신호 : TTL Level
- 기타 : 수출용

OM-1230DC



- 공급처 : 오리온 전기
- 전화 : (02) 784-8755
- 크기 : 12인치
- 수직주파수 : 50 / 60Hz
- 수평주파수 : 15.75 / 18.43kHz
- 입력신호 : TTL Level
- 기타 : 수출용

OM-123C

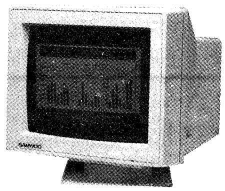


- 공급처 : 오리온 전기
- 전화 : (02) 784-8755
- 크기 : 12인치
- 수직주파수 : 60Hz
- 수평주파수 : 15.75kHz
- 입력신호 : Composite
- 기타 : 수출용

MT-12

- 공급처 : 한국전자
- 전화 : (02) 757-5700
- 크기 : 12인치
- 해상도 : 640×200 / 720×350 도트
- 수직주파수 : 60 / 50Hz
- 수평주파수 : 15.75kHz / 18.43kHz
- 입력신호 : TTL
- 기타 : 수출용

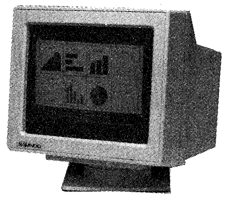
DB-430T



- 공급처 : 삼우 트레이딩
- 전화 : (02) 745-0301
- 크기 : 14인치
- 해상도 : 1000라인
- 수직주파수 : 50Hz
- 수평주파수 : 18.43kHz
- 입력신호 : TTL Separate

• 기타 : 수출용

DB-429B

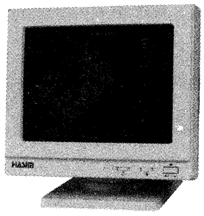


• 공급처 : 삼우 트레이딩
• 전화 : (02) 745-0301
• 크기 : 14인치
• 해상도 : 1000라인
• 수직주파수 : 60Hz
• 수평주파수 : 15.75kHz
• 입력신호 : TTL Separate

DM-216

• 공급처 : 삼우 트레이딩
• 전화 : (02) 745-0301
• 크기 : 12인치
• 해상도 : 1000라인
• 수직주파수 : 60Hz
• 수평주파수 : 15.75kHz
• 입력신호 : Composite

MCF-142W



• 공급처 : 하심전자
• 전화 : (02) 335-2837
• 크기 : 14인치
• 해상도 : 720×350도트
• 수직주파수 : 50Hz
• 수평주파수 : 18.432kHz
• 입력신호 : TTL level positive

MCF-241W

• 공급처 : 하심전자
• 전화 : (02) 335-2837
• 크기 : 14인치
• 해상도 : 720×480도트
• 수직주파수 : 50 / 60 / 70Hz
• 수평주파수 : 31.5kHz
• 입력신호 : 아날로그 신호

컬러 모니터

ECM1911

• 공급처 : 어플라이드 엔지니어링
• 전화 : (02) 679-7691
• 해상도 : 1024×800도트
• 도트피치 : 0.31㎜
• 수직주파수 : 45 / 86Hz
• 수평주파수 : 15kHz / 34kHz
• 입력신호 : 아날로그 RGB / TTL RGB겸용
• 주사방식 : interlace와 non-interlace 겸용
• 가격 : 350만원(VAT 별도)

CM2086A

• 공급처 : 어플라이드 엔지니어링
• 전화 : (02) 679-7691
• 해상도 : 1280×1024도트
• 도트피치 : 0.31㎜
• 수직주파수 : 52Hz
• 수평주파수 : 46 / 52kHz
• 입력신호 : 아날로그 RGB
• 주사방식 : non-interlace
• 가격 : 350만원(VAT 별도)

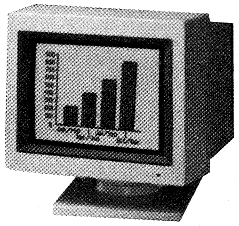
HJ6905K

• 공급처 : 어플라이드 엔지니어링
• 전화 : (02) 679-7691
• 해상도 : 1280×1024도트
• 도트피치 : 0.31㎜
• 수직주파수 : 50 / 80Hz
• 수평주파수 : 40 / 70kHz
• 입력신호 : 아날로그 RGB
• 주사방식 : non-interlace
• 가격 : 350만원(VAT 별도)

GDM-1953

• 공급처 : 어플라이드 엔지니어링
• 전화 : (02) 679-7691
• 해상도 : 1280×1024도트
• 도트피치 : 0.31㎜
• 수직주파수 : 59.98Hz
• 수평주파수 : 63.34kHz
• 입력신호 : 아날로그 RGB
• 주사방식 : non-interlace
• 가격 : 450만원 (VAT 별도)

MA-12



• 공급처 : 한국전자
• 전화 : (02) 757-5700
• 해상도 : 640×480 / 720×400 / 720×350도트
• 수직주파수 : 60Hz / 70Hz
• 수평주파수 : 31.5kHz
• 입력신호 : 아날로그
• 기타 : 수출용

CT-13L

• 공급처 : 한국전자
• 전화 : (02) 757-5700
• 크기 : 14인치
• 해상도 : 640×200도트
• 도트피치 : 0.52㎜
• 수직주파수 : 60Hz
• 수평주파수 : 15.75kHz
• 입력신호 : RGB TTL positive
• 기타 : 수출용

CT-13M

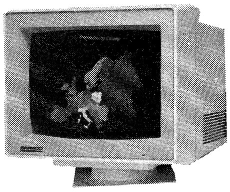
• 공급처 : 한국전자
• 전화 : (02) 757-5700
• 크기 : 14인치
• 해상도 : 640×400도트
• 도트피치 : 0.31㎜
• 수직주파수 : 60Hz

- 수평주파수 : 25kHz
- 입력신호 : TTL positive
- 기타 : 수출용

CT-13

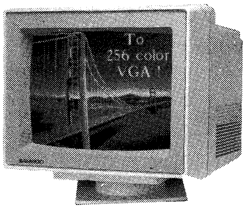
- 공급처 : 한국전자
- 전화 : (02) 757-5700
- 크기 : 14인치
- 해상도 : 640×200도트
- 도트피치 : 0.39㎜
- 수직주파수 : 60Hz
- 수평주파수 : 15.75kHz
- 입력신호 : TTL positive
- 기타 : 수출용

DG442H



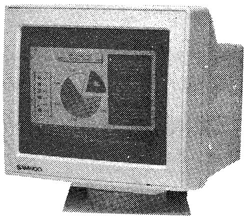
- 공급처 : 삼우 트레이딩
- 전화 : (02) 745-0301
- 크기 : 14인치
- 해상도 : 800×600도트
- 도트피치 : 0.31㎜
- 수직주파수 : 50 / 100Hz
- 수평주파수 : 15 / 38kHz
- 입력신호 : 아날로그 또는 TTL
- 비디오 보드 : MDA, HGC, CGA, EGA, VGA, MCGA
- 기타 : 수출용

DH-441



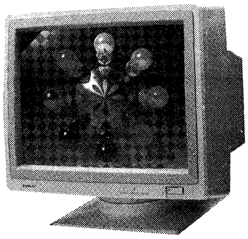
- 공급처 : 삼우 트레이딩
- 전화 : (02) 745-0301
- 해상도 : 720×480도트
- 도트피치 : 0.26㎜
- 수직주파수 : 50 / 70Hz
- 수평주파수 : 31.5kHz
- 입력신호 : TTL level
- 비디오보드 : VGA, MCGA
- 기타 : 수출용

DB-428A



- 공급처 : 삼우 트레이딩
- 전화 : (02) 745-0301
- 크기 : 14인치
- 해상도 : 1400라인
- 수직주파수 : 50 / 70Hz
- 수평주파수 : 31.45kHz
- 입력신호 : 아날로그
- 비디오보드 : VGA, MCGA
- 기타 : 수출용

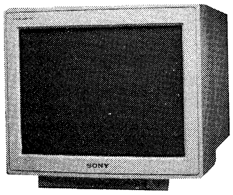
CM2086A



- 공급처 : 다산시스템
- 전화 : (02) 757-7877
- 크기 : 20인치
- 해상도 : 1280×1024도트
- 도트피치 : 0.31㎜
- 수직주파수 : 55 / 65Hz
- 수평주파수 : 46 / 64kHz
- 입력신호 : RGB 아날로그
- 가격 : 330만원(VAT 별도)

GDM-1606

- 공급처 : 삼원테크
- 전화 : (02) 783-0884
- 해상도 : 1280×1024도트

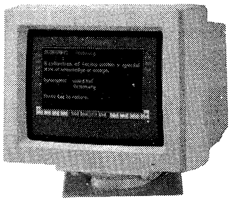


- 도트피치 : 0.26㎜
- 수직주파수 : 59.98Hz
- 수평주파수 : 63.34kHz
- 입력신호 : RGB
- 주사방식 : non-interlace
- 가격 : 190만원(VAT 별도)

GDM-1953

- 공급처 : 삼원테크
- 전화 : (02) 783-0884
- 크기 : 19인치
- 해상도 : 1280×1024도트
- 도트피치 : 0.31㎜
- 수직주파수 : 59.98Hz
- 수평주파수 : 63.34kHz
- 입력신호 : RGB
- 주사방식 : non-interlace
- 가격 : 380만원(VAT 별도)

CA-13



- 공급처 : 한국전자
- 전화 : (02) 757-5700
- 크기 : 14인치
- 해상도 : 640×480 / 720×400 / 720×350도트
- 도트피치 : 0.41㎜
- 수직주파수 : 60 / 70Hz
- 수평주파수 : 31.5kHz
- 입력신호 : 아날로그
- 기타 : 수출용

멀티싱크 모니터

JC-2001VM

- 공급처 : 삼원테크
- 전화 : (02) 783-0884
- 크기 : 20인치
- 해상도 : 1024×768도트
- 수평주파수 : 21.8kHz / 50kHz
- 입력신호 : TTL 아날로그
- 비디오보드 : EGA, MCGA, VGA
- 가격 : 300만원(VAT 별도)

JC-1501VM

- 공급처 : 삼원테크
- 전화 : (02) 783-0884
- 크기 : 15인치
- 해상도 : 960×720도트
- 도트피치 : 0.31㎜
- 수직주파수 : 50 / 80Hz
- 수평주파수 : 21.8kHz / 45kHz
- 입력신호 : TTL / 아날로그
- 비디오보드 : EGA, MCGA, VGA
- 가격 : 130만원(VAT 별도)

CM2085M

- 공급처 : 삼원테크
- 전화 : (2) 783-0884
- 크기 : 20인치
- 해상도 : 1280×1024도트
- 도트피치 : 0.31㎜
- 수직주파수 : 50 / 100Hz
- 수평주파수 : 30 / 64kHz
- 입력신호 : RGB 아날로그
- 가격 : 330만원(VAT 별도)

JC-1402HM

- 공급처 : 삼원테크
- 전화 : (02) 783-0884
- 크기 : 14인치
- 해상도 : 800×560도트
- 도트피치 : 0.31㎜
- 수직주파수 : 50 / 80Hz
- 수평주파수 : 15.5kHz / 35kHz
- 입력신호 : TTL 아날로그
- 비디오보드 : CGA, 허큘리스, MDA, EGA, VGA
- 가격 : 85만원(VAT 별도)

Multi-14C

- 공급처 : 하심전자
- 전화 : (02) 335-2837
- 크기 : 14인치
- 해상도 : 800×560도트
- 도트피치 : 0.31㎜
- 수직주파수 : 56 / 62Hz
- 수평주파수 : 15.5kHz / 35kHz
- 입력신호 : TTL Level
- 비디오보드 : CGA, EGA, PGA

메모리 할당과 할당 해제

MS-DOS에서의 메모리 운용기법은 예전부터 중요한 문제로 대두되고 있다. 여기서는 이러한 메모리의 운용기법중 메모리할당과 할당메모리 해제기능에 대해 알아보자.

기계어에서 이 두 기능은 마이크로프로세서의 AH레지스터와 인터럽트 21H 기능번호에 의해 호출되고 이들의 기능번호는 메모리할당의 경우가 48H이고, 할당메모리 해제의 경우가 49H이다.

부가적으로 이들 48H와 49H에 대해 알아 보면 할당기능은 1패러그래프가 16바이트인 할당할 메모리의 패러그래프수를 필요로 하고 이 패러그래프의 수는 마이크로프로세서의 BX 레지스터내에 위치시켜야 한다.

할당메모리 해제의 경우는 해제할 메모리의 세그먼트 어드레스를 필요로 하는데 이것은 ES 레지스터에 위치시켜야 한다.

각 기능이 수행되면 하나의 값으로 리턴되는데 할당기능이 리턴할 때 에러가 발생하지 않으면 AX에 할당된 메모리의 세그먼트 번지가 기록되고 에러가 발생되면 BX에 사용 가능한 최대 패러그래프수가 기록된다. 에러 발생시 캐리비트는 세트(=1)되고 에러 코드는 07H와 08H로 리턴되는데 07H는 메모리의 제어블록을 파괴시킬 경우이고 08H는 메모리가 부족한 경우이다.

할당메모리 해제의 경우는 기능번호 48H에 의해 할당된 메모리를 해제하는데 에러가 발생되지 않으면 아무것도 리턴되지 않고 에러가 발생하면 캐리비트가 세트(=1)된다. 에러코드는 07H나 09H로 리턴된다. 여기서 07H인 경우는 메모리의 제어블록이 파괴되었을 경우이고, 09H는 ES로 지정된 메모리가 할당되지 않았을 경우이다.

리스트 1과 2에 이들 두 기능의 호출을 위해 필요한 어셈블리코드를 보였다. 리스트 3은 베이직에서 이 두 기능을 호출하는 것을 보인 것이다.

할당경로는 처음부터 이용할 수는 없기 때문에 그 자체로

〔리스트 1〕 DOS 메모리 할당

```
                      page 50,132
0000                  alloc segment para
                         assume cs:alloc
                         assume ds:alloc
                         assume es:alloc
0000                  allocate proc far
                      ;
                      ;메모리블록을 할당하는데 DOS 호출을
                      ;인정하는 베이직 루틴.
                      ;ALLOC(MEMORY)호출-베이직이 베이직으로
                      ;리턴된 루틴을 호출할 때,메모리는
                      ;메모리의 할당된 블록의 세그먼트를
                      ;갖는다.
                      ;7과 8은 할당이 실패되었을 때 지정되는
                      ;에러코드이다.
                      ;
0000  55              push bp
0001  8B EC           mov  bp,sp
0003  8B 5E 06        mov  bx,[bp+6]  ;메모리의 번지 전송
0006  8B 1F           mov  bx,[bx]    ;할당된 바이트 번호 전송
0008  B4 48           mov  ah,48h     ;DOS 기능번호
000A  CD 21           int 21h         ;DOS 호출 실행
000C  8B 5E 06        mov  bx,[bp+6]  ;메모리 번지 전송
000F  89 07           mov  [bx],ax    ;메모리내에 할당된 메모리 세그먼트번지를 전송
0011  5D              pop  bp
0012  CA 0002         ret  2
                      ;
                      allocate endp
0015                  alloc ends
                      end
```

는 공간을 할당할 수 없다. 단지 베이직 프로그램상에서 이를 실행해 보기 위해서는 우선 행번호 10에서 보는바와 같이 CLEAR 명령을 사용한다.

한번 할당된 루틴은 행번호 40~60에서 인스톨작업을 하게 된다. 그리고 할당메모리 해제루틴에 대한 할당메모리의 세그먼트 번지를 행번호 70에서 호출한다.

행번호 120에서는 할당된 메모리를 회수한다.

리스트 3의 프로그램을 베이직상에서 입력하고 다음과 같은 명령문으로 실행하는지 확인해 본다.

```
DEF SEG:PRINT PEEK(&HFFDF)
```

85가 출력될 것이다. 여기서 프로그램과 동일한 숫자가 아니라서 놀라는 독자도 있을 것이다. 그러나 출력이 10진수로 표현된다는 것을 염두에 두면 이해가 갈 것이다.

만약 모든 사람이 메모리에 그들의 프로그램을 상주시키는 곳을 결정하기 위해 DOS를 사용한다면 동일하게 상주하는 프로그램이 오버랩되거나 서로 충돌하는 일은 없을 것이다. 그러나 단지 DOS는 알고 있는 데이터나 프로그램을 보호할 수 있다는 사실에 유의해야 한다. PC

〔리스트 2〕 DOS 메모리 할당 해제

```
                page 50,132
0000            dealloc segment para
                    assume cs:dealloc
                    assume ds:dealloc
                    assume es:dealloc
0000            dlc proc far
                ;
                ;ALLOC를 사용하기 이전에 할당된
                ;메모리의 블록을 회수하기 위해 DOS호출을
                ;인정하는 베이직 루틴.
                ;DEALLOC(MEMORY)호출-베이직의 루틴을 호출할 때
                ;메모리는 할당 회수 메모리 블록의 세그먼트번지를
                ;갖는다.
                ;루틴을 베이직으로 리턴할 때 메모리는 원래의
                ;세그먼트번지나 에러코드를 갖는다.
                ;7과 9는 할당이 부족한 경우 리턴된다.
                ;
0000  55        push bp
0001  06        push es
0002  8B EC     mov  bp,sp
0004  8B 5E 06  mov  bx,[bp+6] ;메모리번지 전송
0007  8E 07     mov  es,[bx]   ;할당 회수 메모리의 세그먼트번지 전송
0009  B4 49     mov  ah,49h    ;DOS 기능번호
000B  CD 21     int  21h       ;DOS 호출 실행
000D  8B 5E 06  mov  bx,[bp+6]
0010  89 07     mov  [bx],ax   ;메모리내에 에러코드 전송
0012  07        pop  es
0013  5D        pop  bp
0014  CA 0002   ret  2
                ;
                dlc endp
0017            dealloc ends
                end
```

〔리스트 3〕 베이직에서의 DOS 메모리 기능

```
10 CLEAR,&HFFDF ―――――― 할당 루틴을 위해 베이직상에서 약간의 바이트 준비
20 DEFINT A-Z
30 DEF SEG:ALLOC=&HFFDF:DMEMORY=2:DEALLOC=0
40 RESTORE 50:FOR X=0 TO 20:READ Y:POKE X+ALLOC,Y:NEXT: ―――― 할당 인스톨
50 DATA &h55,&h8b,&hec,&h8b,&h5e,&h06,&h8b,&h1f,&hb4,&h48,&hcd
60 DATA &h21,&h8b,&h5e,&h06,&h89,&h07,&h5d,&hca,&h02,&h0
70 CALL ALLOC(DMEMORY) ―――――― 할당 해제 루틴을 위해 할당 메모리를 DOS로 호출
80 DEF SEG=DMEMORY
90 RESTORE 100:FOR X=0 TO 22:READ Y:POKE X,Y:NEXT: ―――― 할당 해제 인스톨
100 DATA &h55,&h06,&h8b,&hec,&h85,&h5e,&h06,&h8e,&h07,&hb4,&h49,&hcd
110 DATA &h21,&h8b,&h5e,&h06,&h89,&h07,&h07,&h5d,&hca,&h02,&h00
120 CALL DEALLOC(DMEMORY)
130 END
```

●광장●

뭐든지 물어보세요

수치데이터 출력시 앞에 공백이 생기는 이유는

저는 베이직을 사용하고 있는 독자입니다. 수치데이터를 일정한 자릿수로 출력을 하는 프로그램을 작성하는데 숫자 앞의 공백은 '0'으로 채울려고 합니다. 그런데 실제로 출력을 해 보면 숫자앞에서 공백이 한개있는 채로 출력이 되어 버리는데('000540'인 경우 '00 540'으로 출력됨) 아무리 시도를 해보아도 계속 실패하여 이렇게 도움을 요청합니다. 어떻게 하면 이 공백을 없앨 수 있읍니까?

김영석 / 서울시 마포구 아현동 457-37

원인은 수치데이터의 부호부분 때문입니다.

컴퓨터에서는 숫자가 수치형태와 문자형태로 구분하여 처리되고 있읍니다. 보통 프로그램 내에서 처리되고 있을 때는 수치형태로 있다가 출력시킬 때는 편집이 가능하도록 문자형태로 변환시키고 있읍니다. 질문하신 독자의 경우도 출력할 수치앞에 '0'을 첨가했다는 것은 문자형 데이터 상태로 출력한 경우인데 문제는 '0'과 숫자사이에 생기는 공백일 것입니다.

이 공백은 수치데이터의 부호부분이 문자형 데이터로 변환될 때 공백으로 처리되기 때문에 생기는 현상으로 다음의 리스트에서와 같이 문자변환 함수를 이용하면 해결됩니다. 즉 수치데이터에 100000을 더하고 (5자리로 출력할 경우) 문자변환을 하면 공백은 최좌측으로 오게 됩니다. 이 상태에서 오른쪽부터 5바이트를 취하면 공백은 포함되지 않게 됩니다. 이 외에 다른 방법도 가능하지만 기본적인 개념은 비슷합니다.

```
1 CLS
10 INPUT A
20 C=A+100000!
23 B$=STR$(C)
25 C$=RIGHT$(B$,5)
40 PRINT C$
```

복수개의 화일이 실행되지 않습니다.

dbase초보자로 공개된 프로그램 리스트를 그대로 타이핑하여 입력한 후 도트프롬프트 상태에서 실행시켜 보았더니 "TOO MANY FILES OPEN"이라는 에러메시지가 나타납니다. 프로그램을 살펴보아도 잘못된 부분이 없고 다른 프로그램을 입력해도 마찬가지 에러가 발생합니다. 물론 CONFIG.SYS도 존재합니다. 그리고 7비트 완성형 한글로 dbase프로그램을 사용하였는데 한글이 출력되지 않고 엉뚱한 글자가 출력됩니다. 왜 그런지요.

백종학 / 서울시 관악구 신림3동 706-29

CONFIG.SYS는 루트 디렉토리에 있어야 합니다.

위의 에러메시지는 CONFIG.SYS의 내용이 정의되어 있지 않은 상태에서 복수개의 화일을 오픈할 경우에 발생하는 메시지입니다.

CONFIG.SYS의 내용이 정의되어 있다고 하였는데 아마 루트디렉토리에 존재하지 않는 경우이거나 CONFIG.SYS의 내용이 잘못 지정된 경우일 겁니다. dbase는 복수개의 화일을 사용할 수 있도록 구현되어 있읍니다. DBASE Ⅲ+는 최대 15개의 화일을 사용할 수 있으며 다음과 같이 지정하면 됩니다. 그리고 CONFIG.SYS는 반드시 루트디렉토리에 있어야 합니다.

```
C>copy con config.sys
files = 15
buffers =10
^Z
```

그리고 한글이 출력될 부분에 엉뚱한 문자가 출력된다고 하였는데 이것은 7비트완성형 한글카드 자체가 구성될때 자주 사용되지 않는 부분에 한글을 기록시켜 사용하는데, 이것이 영문자 소문자와 같이 위치하고 있어 이런 일 발생합니다. 7비트 완성형 한글 카드자체의 문제점인 관계로 한글 카드를 바꾸지 않는한 다른 방법은 없읍니다. 하지만 몇가지 주의해서 사용하는 방법이 있읍니다. 한글과 같이 사용할 경우는 영문자는 대문자를 사용하고 소문자의 사용을 피하고 대문자와 소문자의 구분을 하지않는 명령이나 옵

션은 사용을 피함으로써 한글부분에 엉뚱한 것이 출력되는 것을 피할 수 있읍니다.

터보파스칼의 SHELL기능이 작동되지 않습니다.

터보파스칼을 사용하는 도중에 문제가 있어 문의드립니다. IBM XT호환 기종으로, 구입한지는 7개월 정도 되었읍니다. 그런데 터보파스칼4.0의 OS SHELL기능을 사용하면 'Memory all ocation error Cannot Start Command exiting Dos error trying to execute A:\ command.com press ESC"라는 메시지가 화면에 출력됩니다. 그 후 다시 메뉴로 돌아와서 Quit기능을 선택하면 "memory allocation error cannot load command system halted"라고 출력됩니다. DOS버전 3.1과 2.1에서는 동작이 되는데 왜 DOS3.2이상의 버전에서는 위와 같은 에러가 발생하는지요.

강형석 / 서울시 송파구 송파동 한양1차 APT 1동 1001호

CONFIG.SYS를 조정하십시오.

이런 문제가 발생하는 것은 강형석씨가 사용하고 있는 DOS3.3 디스켓에 문제가 있기 때문입니다.

터보파스칼 4.0의 OS SHELL기능을 사용할 때 나오는 "memory allocation error cannot start command exiting DOS error trying to execute"라는 메시지는 터보파스칼의 에러가 아니고 DOS에러인 것입니다. 이 에러는 DOS가 코맨드 프로세서의 또다른 명령을 로드하려 할 때 발생하고 또한 Configuration화일의 files 및 Buffer 지정을 잘못 선택했어도 나타나게 됩니다. 그리고 메인화일이 메모리에서 동작되는중 오버레이화일이나 외부 루틴을 끌어들일 때 메모리 용량이 부족할 때도 위와같은 에러가 발생합니다.

Quit명령을 선택했을 때 나타나는 에러메시지는 DOS의 가용메모리 영역이 프로그램에 의해 파괴되었기 때문입니다. 따라서 강형석씨의 경우는 CONFIG. SYS화일을 재조정하면 해결될 것 같습니다. 다음 리스트를 참조하시고 CONFIG.SYS화일을 새로이 작성했을 때 시스템을 다시 재부팅해야 한다는 것을 잊지 마십시오.

```
C>copy con config.sys
files = 15
buffers =10
^Z
C>
```

SUPER-16E기종에서 640KB로 메모리를 확장하려면

저는 XT기종을 사용하고 있는 독자입니다. 모델은 현대에서 보급형으로 내놓은 SUPPER-16E입니다. 그런데 메인 메모리를 256KB에서 640KB로 확장하려면 메모리확장 보드를 사용하여야 한다고 들었는데 확장카드를 사용하지 않고 메모리를 확장할 수 있는 방법은 없는지요 만약 방법이 있다면 알려 주십시오.

김정도 / 부산시 북구 화평동 주공APT 16동 207호

먼저 자기가 가진 기종이 어느 모델인지 알아야

SUPER-16E는 신형과 구형의 두가지 모델이 있읍니다. 따라서 메모리를 확장하려면 먼저 자기가 가진 모델이 어느형에 속하는지를 알아야 합니다. 신형일 경우는 메인 메모리를 최대 640KB까지 확장하는 것이 가능합니다.

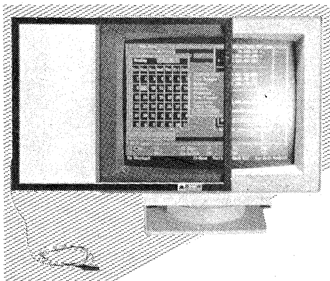
메모리 확장방법은 먼저 SUPPER-16E의 케이스를 열고 보면 메모리확장용 소켓이 12개가 비어 있는 것을 알 수 있읍니다. 이중에 8개의 소켓에 256K 비트램을 꽂아주시면 512KB까지 확장이 되는데 이때 256램의 1번핀 옆에 있는 홈이 기존의 것과 같은 방향으로 일치하도록 꽂아 주셔야 합니다.

그리고 640KB확장시는 512KB까지 확장시킨후 남아 있는 4개의 소켓에 꽂아주신후 점퍼를 조정하시면 됩니다. 점퍼의 조정은 키보드단자 밑부분에 5개의 단자가 있는데 이중 2, 4, 5번이 연결되어 있읍니다. 이 단자중 2번단자를 빼내어 1번단자에 연결하시면 됩니다.

구형모델에서는 메인보드 자체에서는 512KB까지 밖에 확장되지 않읍니다. 640KB까지 확장하려면 별도의 메모리확장카드를 장착하여야 가능합니다. 메모리 확장은 자신이 직접 램을 구입하여 설치하는 것 보다는 구입하신 전문점을 이용하는 것이 바람직합니다.

전문점을 이용할 경우 256KB에서 640KB 까지 확장하는데 9만원정도 소요되는데 확장카드의 경우도 이와비슷한 가격입니다. PC

애독자 선물



3명

폴리에스터에 특수 구리도금과
아크릴 수지로 가공 처리된
첨단기술의 V.D.T 증후군
방지용 컴퓨터 보안기이다.

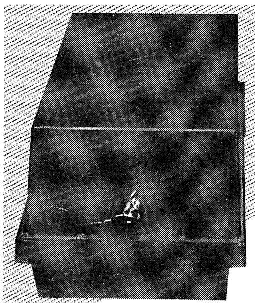
삼종실업 제공
TEL : 784－6931



5명

컴퓨터에 전문지식이 없는 사람도
쉽게 접근할 수 있도록
꾸며진 좋은 지침서이다.

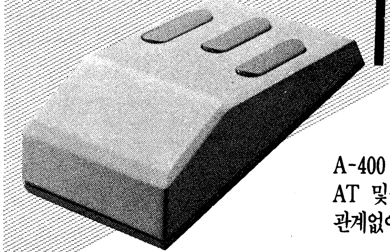
영진출판사 제공
TEL : 719－1119



5명

5.25인치 디스켓을 110장이나 저장할 수
있으며, 자물쇠장치가 부착되어 있어
디스켓을 안전하게 보관할 수 있다.

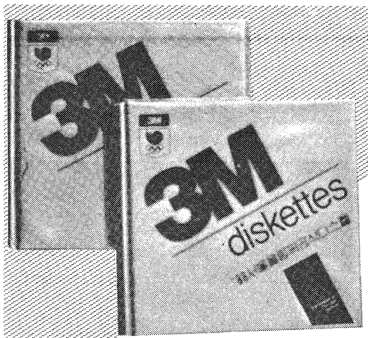
진컴퓨터 제공
TEL : 272－7806



1명

A-400 마우스는 IBM PC/XT/
AT 및 호환기종이면 기종에
관계없이 사용이 가능하다.

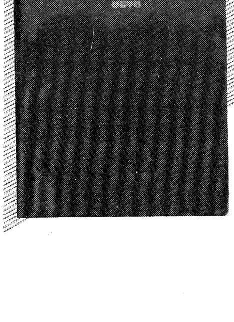
하나컴퓨터 제공
TEL : 717－4894



3명

특수 산화물 코딩 처리된
에러율 0%의 디스켓이다.

한국쓰리엠 제공
TEL : 785－4567



5명

누구나가 한권씩은 지니고 있어야
하는 컴퓨터 용어사전으로 비교적
자세히 설명되어 있다.

오므사 제공
TEL : 754－1044

대잔치



5명

SKC 제공
TEL : 756－5151

균일한 자성체의 배양과 성장으로 도막의 균일도 및 기록신호의 안정도를 극대화하였다.

월간 퍼스널컴퓨터는
독자 여러분의 의견을 소중히 하여
책의 편집에 반영하고자 합니다.
애독자 설문지를 사용하여
질의내용에 성의껏 답하시어
15일까지 보내주시면 27명을 뽑아
선물을 드립니다.

※ 서울 거주자는 4월30일까지 신분증을 지참하시고 직접 본사로 나오시면 해당 상품을 드리며, 지방에 계신 분은 우송해 드리겠습니다. 기간은 4월30일까지이며, 이후는 무효로 처리합니다.

※ 애독자 설문지는 225페이지에 있읍니다.

애독자 선물대잔치
3월호 당첨자 명단

● 컴퓨터 보안기

김진수 대구시 중구 남산2동 573-6
채장국 경남 산청군 단성면 관정리 260

● 디스켓 박스(10장들이)

김성환 전북 군산시 미원동 370-10
이장원 경기도 부천시 중구 원미2동 원미APT 3-301
조영길 서울 노원구 상계6동 주공APT 204-1509
최재신 부산시 사하구 하단동 592-5 18/5
편용헌 인천시 남구 학익1동 613-9 장미APT상가 2-205

● 디스켓 보관함

김동수 서울 서초구 잠원동 한신8차 304동 110호
김순호 부산시 강서구 강동동 1283-8 8/3
서동진 서울 송파구 잠실2동 279동 105호
손재열 충남 대전시 서구 정림동 코스모스APT D-513
이현기 대구시 서구 평리5동 1511-26 29/2

● IBM PC 입문

김동철 부산시 서구 부민동 3가 8번지 12/2
문윤자 경남 삼천포시 선구동 58-12 삼보컴퓨터사무기상사
이희성 부산시 북구 금곡동 1476번지 2/2
장종철 광주시 북구 용봉동 157-33
한인수 서울시 은평구 불광2동 320-13

● 컴퓨터 용어사전

김정걸 서울 강서구 화곡4동 825-18
박기덕 경기도 안산시 고잔2동 주공APT 802-1506
서홍영 전남 여천군 화정면 백야리 화정보건지소
임성록 광주시 동구 동명2동 200-83 12/3
최용진 서울 강동구 둔촌동 주공APT 416-604

● 디스켓 자켓(4 장들이)

강태하 서울 동작구 사당4동 279-36
김현숙 서울 노원구 공릉1동 517-59 10/7
남상희 부산시 동구 수정1동 1043번지
서성식 서울 성북구 길음2동 1272-296
박현상 경기도 과천시 부림동 주공APT 714-101
유상호 전남 곡성군 곡성읍 763번지
이호영 경기도 시흥군 소래읍 신천리 37-6
정수미 대전시 중구 대흥동 현대APT 1-1203
함형준 서울 구로구 구로동 142-58 10/3
현혜숙 제주도 제주시 아라2동 966번지

책을 마감하고

●봄이 왔네요.
새봄맞이 대청소도
뭇 남성, 여성의 봄바람도
밥맛, 입맛 없는 것도
졸립다는 것도
이래 봄탓, 저래 봄탓
이런 탓에 봄은 빨리 지나쳐 가나봐요.
P·C친구들!
우리는 마음으로 봄을 느껴봐요.〈賢〉

●평등과 획일이란 개념이 있습니다.
굳이 둘의 공통점을 찾자면 '같게 한다'는 것일테지요.
한치높이의 화살표와 반치높이의 화살표가 있다고 합시다. 이것을
한치높이에 맞추는 것은 평등이며
반치높이에 맞추는 것이 획일아닙니까.
그것이 평등과 획일의 차이입니다.〈경〉

●믿음을 갖고 싶으면 옳다고 믿는대로 실천하라는 말이 있듯이, 성실성을 바탕으로 하지 않은 사회생활은 사상누각과 같다고 믿고 나 또한 그런 자세로 살려고 열심히 노력했건만 이상기온에 겨울을 온통 빼앗겼던 것처럼 나의 믿음의 나무엔 온갖 실망들만 화살이 되어 꽂히고, 보람을 송두리째 앗아갔던 지난겨울. 몹시 아팠고 힘들었지만 겨울이 가고 새봄이 온것처럼 나의 믿음의 나무가 아무리 거센 폭풍우에도 흔들리지 않고 좀더 풍성한 보람과 희망의 열매가 열릴 수 있는, 굳건한 뿌리를 내리도록 가꾸어 보련다.〈七〉

●덩덩 덩더꿍 덩덩 덩더꿍
잡귀는 썩 물러가라
예전부터 자주왔던 때놈귀신
징글징글한 왜놈귀신
온갖 잡귀들은 물러가라
쉬~
덩덩 덩더꿍 덩덩 덩더꿍
물러가라 물러가라
온갖 잡귀신 모두 물러가라
덩덩 덩더꿍 덩덩 덩더꿍
아직도 안물러간 귀신이 있구나
젊은놈 젊은년에게 흠뻑씌운 미국귀신
썩 물러가라
덩덩 덩더꿍 덩덩 덩더꿍
물러가라 물러가라
양귀신은 물러가라
쉬~
온갖 잡귀신 모두 물러가고
천하대장군 지하여장군 8도귀신 모두 모여
귀신끼리 흐느러지게 놀아보자.
덩덩 덩더꿍 덩덩…〈石心〉

●얼마전 텔리빌딩에 관한 문제가 지면상으로 발표되어 통신관련 S/W 몇몇업체에서는 민감한 반응을 보였으나 주최측에서는 자금출처문제로 고심을 해 일의 진척이 늦어지는 눈치이다. 거액이 투자되는 사업이다보니 당연히 오랜동안의 관철이 필요하겠지만 하루 빨리 많은 양분을 섭취해 크고 싶어 하는 S/W업체를 더 목마르게 하지 마시고 빠른 시일내에 일이 진척된다면 얼마나 기쁘겠습니까. 텔리빌딩이 완공된 후에도 영세한 S/W업체에게는 그림에 떡이 안된다면 그 또한 얼마나 기쁜일입니까.
〈김현〉

●새악시 볼에 살포시 젖는 봄햇살이 눈을 자극하여 아가씨의 마음을 혼란스럽게 하는 봄입니다. 빼앗긴 우리의 시간에도 봄은 왔기에 시간의 꽃을 찾아 헤매는 모모가 더욱 그리워지는 때입니다.
하나 둘씩 자신의 자리를 찾아가는 사람들의 모습속에서 살아있다는 생동감을 느낍니다.
4월과 함께 허니문으로 직행하는 친구들이 있습니다. 그러한 친구들이 동성이 아닌 이성이라는데 약간의 배반감(?)과 아쉬움을 가지지만 같이 동문수학했던 동창이고 보니 축하를 아니할 수 없답니다. 부디 결혼은 지옥으로 통한다는 유언비어를 소탕할 수 있도록 행복으로 통하는 열쇠를 함께 쥐고 가는 길이기를 빕니다. 그리고 머지 않을 날에(과연 그럴 수 있을까?) 빵빠레 울릴 나의 행복한 날에도 잊지않고 축하해 줄 수 있는 우정을 계속 그리고 변함없이 쭉 함께 할 수 있는 친구로 남아있기를 기대합니다.〈둘리〉

●다섯!
하늘은 푸르고
창가엔
적당한 햇빛이 있어 좋다.
대학캠퍼스에
첫 발을 내디딘 동생에게도
이 모든 기쁨이 가득하길…〈榮〉

●안개꽃이 핀
안개꽃 마을은 송두리째 폭탄속에
박혀버린 세월이고,
흔적도 없이 사라져버린 마을이다.
봄
첫 만남을
활활 타오르는
분노로 맞이하는
안개꽃 마을사람들.
하늘까지 연기로 가리운 채
온 산을 까맣게 물들이고
또 다시
파릇파릇 돋아나는 새 생명을
잉태한 안개꽃 마을은
여름을 기다린다.〈鎔〉